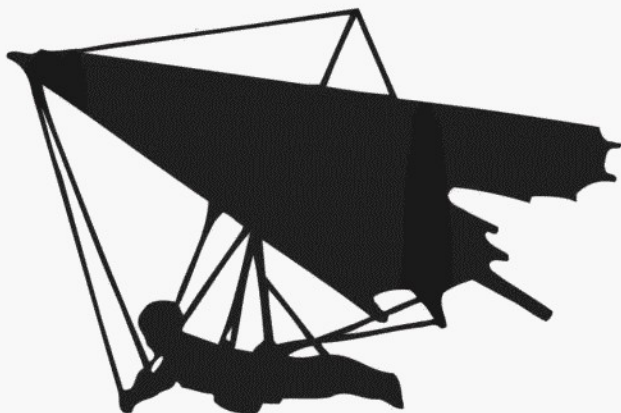


畅销书《HTML 5与CSS 3权威指南》姊妹篇，HTML 5实战进阶必备

注重实战：包含28个中大型案例，每个案例既有直观的效果图，又有详尽的源代码分析  
讲解透彻：对每个案例所涉及的理论精要进行深入剖析，理论与实践完美结合



陆凌牛 著

*Essential HTML 5 and Cases Study*

# HTML 5开发精要 与实例详解



机械工业出版社  
China Machine Press

随着互联网的发展和用户需求的不断变化,从互联网产品的功能和丰富性上来讲,传统的Web开发技术越来越难满足用户的需求。HTML 5的出现,既是技术趋势的发展使然,也是时代需求的必然。

HTML 5的出现和被热捧给开发者提供了一个很好的机会,如果能尽早投身HTML 5变革的热潮,必定能领先一步。如果我们对HTML 5略有了解,但是缺乏实际应用经验,那么本书将是目前的最佳选择(没有之一)。它以HTML 5的理论知识体系为依托,为所有重要的知识点都设计了翔实的案例。这些案例不仅能帮助我们迅速掌握HTML 5的使用方法,而且很多案例都能直接在实际开发中使用。除此之外,还对每个案例涉及的理论知识点进行了深入讲解,可以帮助我们巩固和提高已经掌握的理论知识。

—— HTML 5用户组

本书是以实践方式深入学习HTML 5不可多得的参考书之一。与目前市场上以系统讲解HTML 5理论知识点为主的书不同,它以案例为导向,用28个中大型案例对HTML 5中的重要知识点进行了充分阐释。每个案例都以迭代的方式实现,不仅包含案例需求描述、实现效果展示、开发步骤演示,而且还对案例的源代码进行了分析和点评,既方便读者重新实现这些案例,又能让读者领会这些案例的原理和所用到的理论知识。除此之外,本书的案例源代码都是作者亲自编写的,都能成功运行,可靠性极高。强烈推荐!

—— HTML51 ( www.html51.com )

本书的姊妹篇《HTML 5与CSS 3权威指南》凭借其内容的系统性、翔实性和实战性获得了良好的口碑,在51CTO等技术社区被广泛关注和讨论,被读者誉为“系统学习HTML 5和CSS 3的最佳选择之一”。如果说《HTML 5与CSS 3权威指南》适合入门和开发时备查,那么本书则适合在此基础上进阶提高,适合掌握了理论知识的读者动手去实践,在实践中巩固已学的理论知识,从而达到在实际开发中灵活运用目的。无论是案例的实战性上,还是关键理论知识的深刻性上,本书都值得看一看!

—— 51CTO ( www.51cto.com )

客服热线: (010) 88378991, 88361066  
 购书热线: (010) 68326294, 88379649, 68995259  
 投稿热线: (010) 88379604  
 读者信箱: hzsj@hzbook.com

华章网站 <http://www.hzbook.com>

网上购书: [www.china-pub.com](http://www.china-pub.com)



上架指导: 程序设计/Web开发

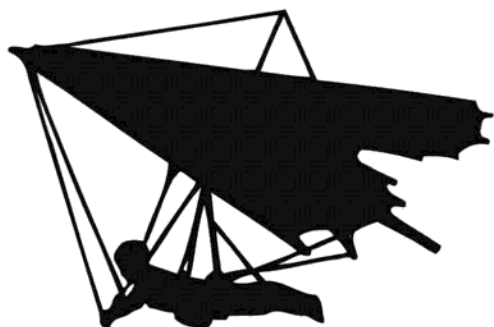
ISBN 978-7-111-36112-1



9 787111 361121

定价: 79.00元





*Essential HTML 5 and Cases Study*

# HTML 5开发精要 与实例详解

陆凌牛 著



机械工业出版社  
China Machine Press

这是一本以综合性案例为导向并辅之以精要知识点讲解的 HTML 5 实战教程。内容分为两大部分：第一部分通过一系列中大型案例全方位对 HTML 5 的各个重要知识点进行了详细的讲解，每个案例包含案例概述、页面效果展示、案例所涉及主要知识点（精要）、源代码剖析 4 个部分，读者既能根据书中的步骤动手实践，又能重点学习案例中用到的核心理论知识，同时还能领会源代码的设计思路和方法；第二部分讲解了 jWebSocket、RGraph、WebGL 等 3 个重要框架和技术的详细使用方法。

全书一共 12 章：第 1 章分别用两个案例演示了如何利用 HTML 5 中的结构元素来构建一个博客网站和企业门户网站；第 2 章用两个案例讲解了表单在 HTML 5 中的使用；第 3 章用 6 个案例讲解了如何利用 Canvas 元素来绘制图形、图像和制作动画；第 4 章用两个案例介绍了文件 API 和拖放 API 的使用方法；第 5 章用 4 个案例讲解了如何打造自己的网页视频播放器、网页音频播放器，以及实现视频实时回放和视频截图等多媒体功能；第 6 章用 6 个案例全面讲解了 HTML 5 中的本地存储技术；第 7 章用单点登录和获取批量数据这两个案例讲解了 HTML 5 中的跨文档的消息传输技术；第 8 章用两个案例讲解了如何利用 Web Workers 实现多线程处理；第 9 章用一个案例讲解了如何利用 Geolocation API 来获取地理位置信息；第 10 ~ 13 章分别讲解了 Socket 通信框架 jWebSocket、统计图制作插件 RGraph、三维 Web 开发技术 WebGL 的详细使用方法，并辅之以丰富的案例。

本书所有案例的源代码都是作者亲自编写并调试和运行成功的。读者可以利用这些代码进行实战练习，也可以根据需要对这些代码进行修改，以观察不同的效果，从而加深对案例代码和书中知识点的理解。

封底无防伪标均为盗版

版权所有，侵权必究

本书法律顾问 北京市展达律师事务所

## 图书在版编目（CIP）数据

HTML 5 开发精要与实例详解 / 陆凌牛著. —北京：机械工业出版社，2011.11

ISBN 978-7-111-36112-1

I. H… II. 陆… III. 超文本标记语言，HTML 5 —程序设计 IV. TP312

中国版本图书馆 CIP 数据核字（2011）第 208142 号

机械工业出版社（北京市西城区百万庄大街 22 号 邮政编码 100037）

责任编辑：姜 影

北京京师印务有限公司印刷

2012 年 1 月第 1 版第 1 次印刷

186mm×240mm·35.25 印张

标准书号：ISBN 978-7-111-36112-1

定价：79.00 元

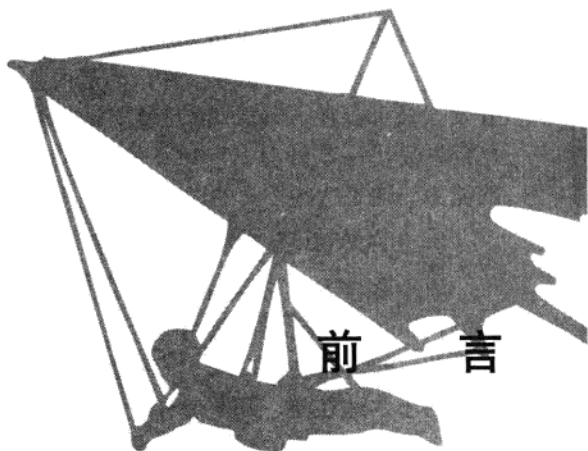
凡购本书，如有缺页、倒页、脱页，由本社发行部调换

客服热线：（010）88378991；88361066

购书热线：（010）68326294；88379649；68995259

投稿热线：（010）88379604

读者信箱：hzjsj@hzbook.com



## 为什么要写这本书

虽然 HTML 5 的最终标准还未尘埃落定，但是它正在孕育一场巨大的革命。从技术的角度来讲，HTML 5 的出现将是 Web 开发标准的一次重要飞跃，它不仅能提供更加丰富、强大、炫丽的功能，还将改变互联网的很多方面。HTML 5 不会完全取代插件，但的确可以使浏览器无需借助插件就能够实现更多的功能——从位置跟踪到把数据保存到云端。HTML 5 的标签将取代那些完成简单任务的插件，而且在某些时候可以开发一些高级的应用。

准确地说，HTML 5 将从以下几个方面改变未来的整个互联网世界。

### 1. 降低插件的重要性

过去，很多功能只能通过插件或者复杂的 hack（本地绘图 API 和本地 Socket 等）来实现，而 HTML 5 提供了对这些功能的原生支持。插件的方式存在很多问题：

- ❑ 插件安装可能失败。
- ❑ 插件可以被禁用或屏蔽（例如 Apple 的 iPad 就不支持 Flash 插件）。
- ❑ 插件自身会成为被攻击的对象。
- ❑ 插件不容易与 HTML 文档的其他部分集成（因为插件边界、剪裁和透明度问题）。

HTML 5 解决这些问题的一个办法就是使浏览器原生地支持相关的标签。只需浏览器支持相应的 HTML 5 标签，如 `<audio>` 和 `<video>` 等各种标签，不需要任何插件，就可以像使用 `<img>` 一样方便地在页面内嵌入音频和视频。



## 2. 支持动态生成图像

过去, 网页中显示的图像来自于直接下载的 GIF 或 JPG 图像, 而在 HTML 5 中, 图像可能并不是直接来自图像文件, 而是由某个 canvas (画布) 对象临时生成的。利用 HTML 5, 开发人员可以在 JavaScript 层针对动态数据进行计算后绘制出各种复杂的图形。另外, 目前已经开发出一些针对 HTML 5 的类似工具, 这些工具将进一步提高 Web 开发人员驾驭图像的能力, 随着这些工具的成熟, 开发人员也将开发出更多更为专业的复杂图形。

## 3. 允许 Web 程序利用本地存储

Web 程序其实早就可以利用浏览器端的本地存储空间存储很多信息, 比如 IE 允许最多 300 个 Cookie, 最多存储 4096 字节的内容。不过, 要开发真正实用的 Web 应用, 需要更多的存储空间。通过 HTML 5, 可以实现这种需求。

对于本地存储, 开发者可以按照需求随意使用, 比如把云服务的应用和数据保存在本地硬盘上, 使云应用的交付、安装和部署都非常像传统的应用程序。无论是否连接到互联网, 云应用程序都可以照常运行, 因为之前已经从服务器上下载了 HTML 5 应用的 JavaScript 代码, 这部分代码就保存在本地。本地数据库实际上扮演的是智能缓存的角色。

对于游戏开发, 可以在本地存储一些情景信息和装备信息, 这样可避免每次与服务器建立连接时都要下载这些信息, 从而节省下载资料的时间。

## 4. 简化 Web 开发的数据提取

从网页中提取过数据的 Web 开发人员都知道, HTML 5 之前的 HTML 结构除了告诉浏览器数据在哪里之外, 几乎不能再提供任何有意义的信息。开发人员需要了解与数据本身有关的信息, 以了解这些数据的真正含义。HTML 5 中所谓的微格式 (microformat) 引入了一种新的机制, 它在 HTML 中新增了一些专门的标签, 以帮助程序员分析标签中数据的真实含义。

没有人能够预测微格式到底会给网络带来多少改变, 但很容易看出, 这种新的机制将给 Web 开发人员带来很多方便, 帮助他们开发出更有效率的 Web 应用。如果有一个好的、标准的方式来表示日期和时间, 那么在为网站开发与时间有关的 Web 程序时, 就无需另外编写专门的代码来分析网站访问者可能使用什么时间格式, 这样, 日历、时间表和日程安排等需要从多个数据源收集时间信息的应用也就变得非常简单了。

## 5. 支持位置服务

在 Web 世界里, 很多人只知道其 IP 地址, 根本不知道那些数字对应的计算机所处的真实世界是什么样的。比如, 过去几乎不可能知道某台计算机所在的地理位置, 而现在通过地理位置服务可以很好地解决这个问题。HTML 5 允许 JavaScript 询问浏览器用户的地理位置, 比如纬度和经度信息。通常桌面系统不支持这一功能 (因为需要有 GPS 或 Wi-Fi), 如果终端是智能手机, 这个功能就可以发挥作用。

## 6. 让 Web 视频播放更流畅

HTML 5 中的 video 元素使 Web 开发人员很容易把视频内容与网页中的其他内容整合起

来,同时也使互联网视频内容越来越丰富,从而使网页成为视频内容的主要发布场所。

毫无疑问,在未来的互联网世界里,HTML 5 将对 Web 开发起到很重要的作用。目前,HTML 5 引起了很多 Web 开发人员的强烈兴趣,越来越多的开发人员开始学习并在实际工作中尝试使用 HTML 5,但是目前市场上的相关图书都是以介绍基础理论知识为主,从纯粹实践的角度去讲解 HTML 5 的书寥寥无几。为了使读者通过实践掌握 HTML 5,笔者对 HTML 5 进行了详细的研究,并收集了一些比较具有代表性的贴近实际工作的案例,在此基础上撰写了本书。希望本书能够为致力于利用 HTML 5 开发 Web 应用的读者提供一些参考,使读者对 HTML 5 有一个比较深入的了解,并能够在未来的互联网开发工作中充分运用这些知识。

## 本书特色

为了使有一定基础的读者能通过实战的方式进一步学习 HTML 5 的相关知识,本书以实例为主,同时对每个实例所涉及的重要理论知识进行了精辟的讲解。总体而言,本书有如下几个方面的特色。

### 1. 案例丰富,实践性强

目前 HTML 5 的应用在国内还处于初级阶段,市面上已经出版的相关图书几乎都以讲解 HTML 5 的功能和特点为主,也就是以理论为主,从实践的角度去写的书很少,涉及大型实际案例的更是少之又少。HTML 5 的功能丰富而强大,掌握了它的基本功能并不代表能自如地将其应用到实际工作中,因为还缺乏一个实战练习的过程。为了弥补上述不足,本书以案例为主,针对 HTML 5 的每个重要知识点都设计了大型的案例,这些案例不仅实践性极强,而且可以直接在工作中使用。

### 2. 内容详细,讲解透彻

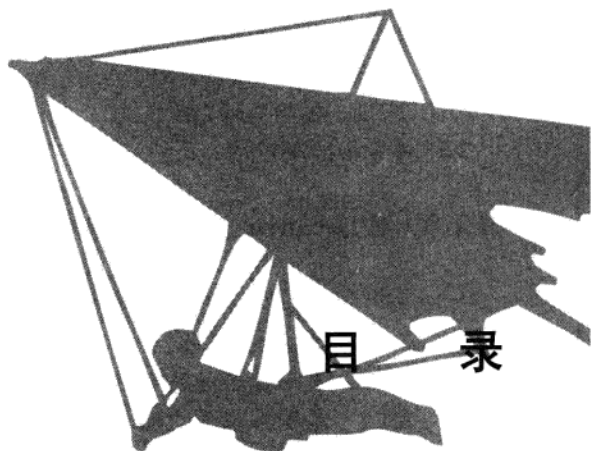
本书的内容非常详细,每个案例的制作步骤都以通俗易懂的语言进行阐述。大部分案例的介绍都按照案例概述、页面显示效果、案例知识点以及代码剖析的顺序,读者在阅读每一个案例时,先获得一个总体印象,然后直观地看到该案例在浏览器中的显示效果,接下来了解案例中所用到的 HTML 5 的相关知识点,最后通过代码剖析来掌握如何一步步实现 HTML 5 的某个功能。通过这样一个循序渐进的方式,读者能够更加深入地了解 HTML 5 的每一个功能点。

### 3. 图文并茂,效果直观

每个案例都配有相应的页面显示效果图,读者在阅读案例代码前可以先通过案例的页面显示效果对案例有个直观的印象,初步了解案例要实现的功能。

### 4. 源码详尽,便于调试

本书所有案例的代码都是笔者亲自调试并运行成功的,读者可以对这些代码进行修改,以观察各种不同效果,并加深对案例代码的理解。



## 目 录

### 前言

## 第 1 章 用 HTML 5 中的结构元素构建网站 /1

- 1.1 案例 1：用 HTML 5 中的结构元素构建一个博客网站 /2
  - 1.1.1 案例知识点 /2
  - 1.1.2 博客首页的实现 /11
  - 1.1.3 文章显示页面的实现 /30
- 1.2 案例 2：用 HTML 5 中的结构元素构建一个企业网站 /39
  - 1.2.1 首页的实现 /39
  - 1.2.2 联系方式页面的实现 /52
- 1.3 本章小结 /55

## 第 2 章 在 Web 表单中使用 HTML 5/56

- 2.1 案例 3：用结构元素制作 Web 应用程序中的菜单 /57
  - 2.1.1 页面显示效果 /57
  - 2.1.2 代码剖析 /58
- 2.2 案例 4：综合运用 HTML 5、jQuery 与 ASP.NET 构建 Web 应用程序 /61



- 2.2.1 案例概述 /61
- 2.2.2 页面显示效果 /61
- 2.2.3 案例知识点 /63
- 2.2.4 代码剖析 /70
- 2.3 本章小结 /96

### 第3章 使用 canvas 元素绘制图形、图像与动画 /97

- 3.1 案例5：使用 canvas 元素绘制美丽的花朵 /98
  - 3.1.1 案例概述 /98
  - 3.1.2 页面显示效果 /98
  - 3.1.3 案例知识点 /100
  - 3.1.4 代码剖析 /103
- 3.2 案例6：使用 canvas 元素绘制指针式动画时钟 /106
  - 3.2.1 案例概述 /106
  - 3.2.2 页面显示效果 /106
  - 3.2.3 案例知识点 /107
  - 3.2.4 代码剖析 /108
- 3.3 案例7：使用 canvas 元素制作简单小游戏 /113
  - 3.3.1 案例概述 /113
  - 3.3.2 页面显示效果 /113
  - 3.3.3 案例知识点 /115
  - 3.3.4 代码剖析 /116
- 3.4 案例8：使用 canvas 元素绘制图像放大镜 /119
  - 3.4.1 案例概述 /120
  - 3.4.2 页面显示效果 /120
  - 3.4.3 案例知识点 /120
  - 3.4.4 代码剖析 /121
- 3.5 案例9：用动画的形式装载图像 /125
  - 3.5.1 案例概述 /125
  - 3.5.2 页面显示效果 /125
  - 3.5.3 案例知识点 /128
  - 3.5.4 代码剖析 /130
- 3.6 案例10：将彩色照片转换成黑白照片 /138
  - 3.6.1 案例概述 /138
  - 3.6.2 页面显示效果 /138

- 3.6.3 案例知识点 /140
- 3.6.4 代码剖析 /141
- 3.7 本章小结 /143

## 第4章 文件 API 与拖放 API/145

- 4.1 案例 11：在浏览器中预览客户端文件并上传 /146
  - 4.1.1 案例概述 /146
  - 4.1.2 页面显示效果 /146
  - 4.1.3 案例知识点 /149
  - 4.1.4 代码剖析 /151
- 4.2 案例 12：使用 Canvas API、文件 API 与拖放 API 制作拼图游戏 /158
  - 4.2.1 案例概述 /158
  - 4.2.2 页面显示效果 /159
  - 4.2.3 案例知识点 /162
  - 4.2.4 代码剖析 /165
- 4.3 本章小结 /176

## 第5章 多媒体播放 /177

- 5.1 案例 13：打造自己的网页视频播放器 /178
  - 5.1.1 案例概述 /178
  - 5.1.2 页面显示效果 /178
  - 5.1.3 案例知识点 /181
  - 5.1.4 代码剖析 /183
- 5.2 案例 14：对视频使用实时回放功能 /191
  - 5.2.1 案例概述 /191
  - 5.2.2 页面显示效果 /191
  - 5.2.3 代码剖析 /192
- 5.3 案例 15：对视频使用截图功能 /195
  - 5.3.1 案例概述 /195
  - 5.3.2 页面显示效果 /195
  - 5.3.3 案例知识点 /196
  - 5.3.4 代码剖析 /196
- 5.4 案例 16：打造自己的网页音频播放器 /199
  - 5.4.1 案例概述 /199

- 5.4.2 页面显示效果 /199
- 5.4.3 案例知识点 /200
- 5.4.4 代码剖析 /201
- 5.5 本章小结 /206

## 第6章 本地存储 /207

- 6.1 案例 17：制作 HTML 5 版本的日程提醒簿 /208
  - 6.1.1 案例概述 /208
  - 6.1.2 页面显示效果 /208
  - 6.1.3 案例知识点 /209
  - 6.1.4 代码剖析 /210
- 6.2 案例 18：临时保存页面中的输入内容 /214
  - 6.2.1 案例概述 /214
  - 6.2.2 页面显示效果 /215
  - 6.2.3 案例知识点 /216
  - 6.2.4 代码剖析 /216
- 6.3 案例 19：使用 HTML 5 制作 Web 应用程序的演示版 /218
  - 6.3.1 案例概述 /218
  - 6.3.2 页面显示效果 /218
  - 6.3.3 案例知识点 /219
  - 6.3.4 代码剖析 /222
- 6.4 案例 20：使用客户端 session/232
  - 6.4.1 案例概述 /232
  - 6.4.2 页面展示效果 /232
  - 6.4.3 案例知识点 /234
  - 6.4.4 代码剖析 /235
- 6.5 案例 21：将本地数据库中的数据提交到服务器端 /248
  - 6.5.1 案例概述 /248
  - 6.5.2 页面展示效果 /248
  - 6.5.3 代码剖析 /249
- 6.6 案例 22：制作可以离线使用的日程提醒簿 /256
  - 6.6.1 案例概述 /256
  - 6.6.2 页面显示效果 /256
  - 6.6.3 案例知识点 /257



#### 6.6.4 代码剖析 /260

#### 6.7 本章小结 /261

### 第7章 跨文档消息传输 /262

#### 7.1 案例 23：通过跨文档消息传输功能实现单点登录 /263

##### 7.1.1 案例概述 /263

##### 7.1.2 页面显示效果 /264

##### 7.1.3 案例知识点 /268

##### 7.1.4 代码剖析 /268

#### 7.2 案例 24：通过跨文档消息传输功能获取批量数据 /300

##### 7.2.1 案例概述 /300

##### 7.2.2 页面显示效果 /300

##### 7.2.3 代码剖析 /301

#### 7.3 本章小结 /310

### 第8章 利用 Web Workers 实现多线程处理 /312

#### 8.1 案例 25：在后台线程中实现对数据库的增删查改操作 /313

##### 8.1.1 案例概述 /313

##### 8.1.2 页面显示效果 /313

##### 8.1.3 案例知识点 /315

##### 8.1.4 代码剖析 /316

#### 8.2 案例 26：在后台线程中实现数据的批量插入 /334

##### 8.2.1 案例概述 /334

##### 8.2.2 代码剖析 /334

#### 8.3 本章小结 /339

### 第9章 利用 Geolocation API 获取地理位置信息 /341

#### 9.1 案例 27：显示计算机或移动设备所在地的地图 /342

##### 9.1.1 案例概述 /342

##### 9.1.2 页面显示效果 /342

##### 9.1.3 案例知识点 /343

##### 9.1.4 代码剖析 /346

#### 9.2 本章小结 /349

## 第 10 章 使用 jWebSocket 框架开发 Socket 通信程序 /350

- 10.1 安装与运行 jWebSocket/351
  - 10.1.1 安装 jWebSocket 服务器 /351
  - 10.1.2 在其他服务器环境下运行 jWebSocket 服务器 /352
  - 10.1.3 将 jWebSocket 服务器作为 Windows 的可执行文件 /354
  - 10.1.4 将 jWebSocket 服务器作为 Windows 的服务 /355
  - 10.1.5 jWebSocket Web 客户端 /355
- 10.2 创建第一个利用 jWebSocket 进行通信的 Web 页面 /357
- 10.3 创建 jWebSocket 服务器端的侦听器 /365
  - 10.3.1 jWebSocket 的通信架构 /365
  - 10.3.2 创建侦听器 /366
- 10.4 jWebSocket 中的令牌 /374
  - 10.4.1 令牌的基本概念 /375
  - 10.4.2 系统令牌 /376
- 10.5 jWebSocket 中服务器端的插件 /383
  - 10.5.1 服务器端插件的基础知识 /383
  - 10.5.2 创建自定义服务器端插件 /385
- 10.6 jWebSocket 中的通道 /393
- 10.7 案例 28：利用 jWebSocket 服务器创建简单聊天室 /398
  - 10.7.1 案例概述 /398
  - 10.7.2 页面显示效果 /398
  - 10.7.3 代码剖析 /401
- 10.8 本章小结 /412

## 第 11 章 RGraph 统计图制作插件 /414

- 11.1 概述 /415
  - 11.1.1 HTML 5 版统计图插件的优越性 /415
  - 11.1.2 使用 RGraph 插件 /415
  - 11.1.3 使用服务器端数据 /416
- 11.2 绘制统计图时所用到的公共属性 /417
- 11.3 绘制柱状图 /422
  - 11.3.1 绘制柱状图时所用到的属性 /422
  - 11.3.2 示例程序 /425

- 11.3.3 使用 obj.getBar 方法 /426
- 11.3.4 绘制分组柱状图 /428
- 11.3.5 使用上下文菜单 /432
- 11.4 绘制折线图 /435
  - 11.4.1 绘制折线图时所用到的属性 /435
  - 11.4.2 绘制基本折线图 /439
  - 11.4.3 使用 getPoint 方法 /441
  - 11.4.4 在一个折线图中绘制多根折线 /444
  - 11.4.5 绘制范围折线图 /446
  - 11.4.6 在一个折线图中使用左右两根不同统计单位的垂直坐标轴 /448
  - 11.4.7 在一个统计图中绘制柱状图与折线图 /450
  - 11.4.8 绘制动态折线图 /452
- 11.5 绘制饼图 /455
  - 11.5.1 绘制饼图时所用到的属性 /455
  - 11.5.2 示例程序 /455
  - 11.5.3 使用 getSegment 方法 /457
- 11.6 绘制横向柱状图 /461
  - 11.6.1 绘制横向柱状图时所用到的属性 /461
  - 11.6.2 示例程序 /463
  - 11.6.3 绘制分组横向柱状图 /464
- 11.7 绘制雷达图 /466
  - 11.7.1 绘制雷达图时所用到的属性 /466
  - 11.7.2 示例程序 /467
- 11.8 增强用户体验 /468
  - 11.8.1 通过拖曳来缩放统计图的尺寸 /468
  - 11.8.2 制作工具条提示信息 /469
  - 11.8.3 制作上下文菜单 /475
  - 11.8.4 放大统计图 /476
  - 11.8.5 允许用户注解统计图 /481
- 11.9 本章小结 /483

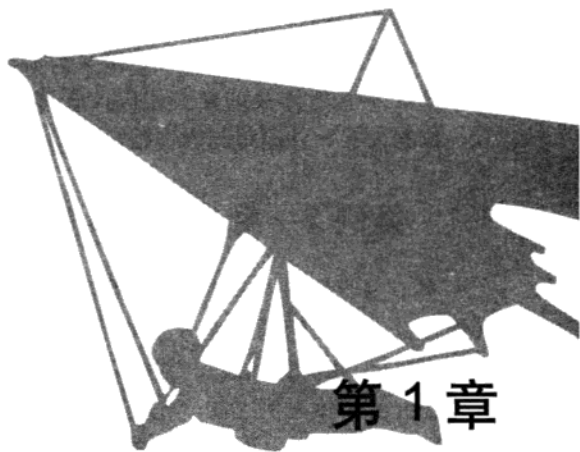
## 第 12 章 使用 WebGL 开发三维图形图像 /484

- 12.1 WebGL 概述 /485
  - 12.1.1 WebGL 的基础知识 /485



- 12.1.2 进行 WebGL 开发之前的准备工作 /485
- 12.2 使用 WebGL 绘制三角形与矩形 /488
  - 12.2.1 下载并使用脚本文件 /488
  - 12.2.2 页面显示效果 /488
  - 12.2.3 代码剖析 /488
- 12.3 使用颜色绘制彩色三角形与矩形 /507
  - 12.3.1 画面式样 /507
  - 12.3.2 代码剖析 /507
- 12.4 制作三维动画 /515
  - 12.4.1 画面式样 /515
  - 12.4.2 代码剖析 /515
- 12.5 制作三维物体 /520
  - 12.5.1 画面式样 /520
  - 12.5.2 代码剖析 /520
- 12.6 使用纹理 /528
  - 12.6.1 画面式样 /528
  - 12.6.2 代码剖析 /528
- 12.7 键盘输入与纹理过滤 /537
  - 12.7.1 画面式样 /537
  - 12.7.2 代码剖析 /538
- 12.8 本章小结 /545

**附录 五大浏览器的最新版对 HTML 5 的支持情况 /546**



## 第 1 章

# 用 HTML 5 中的结构元素构建网站

### 本章内容

- ☐ 案例 1：用 HTML 5 中的结构元素构建一个博客网站
- ☐ 案例 2：用 HTML 5 中的结构元素构建一个企业网站
- ☐ 本章小结

本章主要通过一个博客网站与一个企业网站的制作来向读者展示如何合理运用 HTML 5 中的各种结构元素，搭建出一个语义清晰、结构分明的 Web 3.0 时代的网站。

由于本章所涉及的样式代码比较多，因此只对其中比较重要的样式进行介绍，颜色设置、位置设置、外边距与内边距设置和字体设置等代码均略去。

## 1.1 案例 1：用 HTML 5 中的结构元素构建一个博客网站

本节主要介绍一个使用 HTML 5 中的各种结构元素来构建博客网站的案例，旨在通过该案例使读者触类旁通，充分了解 HTML 5 中的各种结构元素的作用、使用场合及使用方法，从而构建出与之相类似的、结构分明的、语义清晰的 HTML 5 网站。

在介绍具体案例之前，先概要介绍一下 HTML 5 中的网页结构与 HTML 4 中的网页结构的区别；HTML 5 中添加了哪些结构元素，这些结构元素的作用与使用场合是什么；HTML 5 中的网页大纲是什么，这些结构元素会在网页大纲的生成过程中起到什么样的作用，一份网页大纲是根据什么原则生成的。

另外，本节所介绍的知识点均为概要介绍，详细理论知识请参阅笔者所著的《HTML 5 与 CSS 3 权威指南》<sup>①</sup>一书。

### 1.1.1 案例知识点

#### 1. HTML 4 中表达文档结构的方法

一般来说，人们在书写包括 HTML 在内的文档时，习惯上按照类似于“章→节→小节”这样的层次结构来进行。在 HTML 4 网页文档中，为了描述这样的层次结构，一般使用标题元素（h1～h6）。

例如，在 HTML 4 之前，为了在页面中表达“章→节→小节”这样的三层结构，一般采用如代码清单 1-1 所示的代码进行书写。

代码清单 1-1 HTML 4 之前的文档结构的表示方法

---

```
<h1>1 HTML5 的基础知识 </h1>
<h2>1.1 HTML5 概述 </h2>
(1.1 的正文)
<h3>1.1.1 HTML5 是什么? </h3>
(1.1.1 的正文)
<h3>1.1.2 HTML5 中的新增 API</h3>
(1.1.2 的正文)
<h2>1.2 HTML5 快速入门 </h2>
(1.2 的正文)
<h3>1.2.1 HTML 与 XHTML</h3>
(1.2.1 的正文)
```

---

<sup>①</sup> 机械工业出版社于2011年4月出版，已经重印4次，广获读者好评。——编辑注

从代码清单 1-1 所示的文档结构代码中，勉强可以看出该文档的主体结构。但是，在比较复杂的页面中，这种文档结构往往很难看出来。比如对“1 HTML5 的基础知识”来说，它的代码只有“<h1>1 HTML5 的基础知识</h1>”这一行，没有使用其他元素将<h1>元素中的内容包围起来，这一章内容到底从哪儿到哪儿，是很难看出来的。

后来，为了解决这个问题，开发者开始使用 div 元素将一章的内容包围起来，具体使用如代码清单 1-2 所示。

代码清单 1-2 在 HTML 4 中使用 div 元素描述文档结构

---

```
<div >
  <h1>1 HTML5 的基础知识</h1>
  <div >
    <h2>1.1 HTML5 概述</h2>
    (1.1 的正文)
    <div >
      <h3>1.1.1 HTML5 是什么？</h3>
      (1.1.1 的正文)
    </div>
    <div >
      <h3>1.1.2 HTML5 中的新增 API</h3>
      (1.1.2 的正文)
    </div>
  </div>
  <div >
    <h2>1.2 HTML5 快速入门</h2>
    (1.2 的正文)
    <div >
      <h3>1.2.1 HTML 与 XHTML</h3>
      (1.2.1 的正文)
    </div>
  </div>
</div>
```

---

使用 div 元素，这段文档的结构就一目了然了。

但是，最初设计 div 元素的目的并不是为了区分文档的结构，而是为了使用样式。因此，从语义上来说，div 元素是不具备任何语义的，换句话说，该元素是不适合用来划分页面文档结构的。

随着页面文档的不断复杂化，广大开发者发现，如果整个页面都依靠 div 元素来划分文档结构，对于含有大量用来划分文档结构的 div 元素和大量用来使用样式的 div 元素的一个页面，不仔细查看页面代码与样式代码，很难看出整个页面的文档结构。

## 2. HTML 5 中的新增结构元素

在 HTML 5 中，为了使文档结构更加清晰，更容易阅读，增加了很多具有语义性的专门用来划分文档结构的结构元素。接下来，我们对这些结构元素进行概要介绍。

## 4 ❖ HTML 5 开发精要与实例详解

### □ q section 元素

section 元素是对页面文档结构进行划分的最基本也是最主要的结构元素，主要用来对网站或应用程序中的页面上的内容进行层次结构上的划分。一个 section 元素通常由内容及其标题组成。

例如，针对代码清单 1-2 中的代码，我们可以使用 section 元素进行页面文档结构的划分，如代码清单 1-3 所示。

代码清单 1-3 在 HTML 5 中使用 section 元素描述文档结构

---

```
<section>
  <h1>1 HTML5 的基础知识 </h1>
  <section>
    <h2>1.1 HTML5 概述 </h2>
    (1.1 的正文)
    <section>
      <h3>1.1.1 HTML5 是什么? </h3>
      (1.1.1 的正文)
    </section>
    <section>
      <h3>1.1.2 HTML5 中的新增 API</h3>
      (1.1.2 的正文)
    </section>
  </section>
  <section>
    <h2>1.2 HTML5 快速入门 </h2>
    (1.2 的正文)
    <section>
      <h3>1.2.1 HTML 与 XHTML</h3>
      (1.2.1 的正文)
    </section>
  </section>
</section>
```

---

使用 section 元素时需要注意以下两点：

- 不要将 section 元素与 div 元素混淆使用。当一个容器需要直接定义样式或通过脚本定义行为时，推荐使用 div 元素而非 section 元素。
- 通常不建议为那些没有标题（h1 ~ h6 元素）的内容使用 section，可以使用 HTML 5 轮廓工具（下载 HTML 5 轮廓工具的网址为“<http://gsnedders.html5.org/outliner/>”）来检查页面中是否有不包含标题部分的 section。如果使用该工具进行检查后，在某个 section 的说明中有“untitled section”（没有标题的 section）文字，这个 section 就可能属于使用不当（nav 元素或 aside 元素没有标题是合理的）。

### □ article 元素

article 元素代表文档、页面或应用程序中的所有“正文”部分，它所描述的内容应该是独立的、完整的、可以独自被外部引用的，可以是一篇博客、一篇报刊中的文章、一篇论坛

帖子、一段用户评论、一个独立的插件，或任何独立于上下文中其他部分的内容。

在一个 section 元素或 article 元素内，应该只有一个标题，如果有两个标题，则第二个标题会被隐式放入一个新的 section 元素之内，如下面的代码所示，h2 元素之后的内容会被隐式放入一个新的 section 元素内。

```
<article>
  <h1> 标题 </h1>
  <!-- 隐式创建出一个新的 section 元素 -->
  <h2> 副标题 </h2>
  正文
</article>
```

#### □ nav 元素

nav 元素是一个可以作为页面导航的链接组，其中的导航元素链接到其他页面或当前页面的其他部分。并不需要将所有的链接组都放进 nav 元素，只需要将主要的、基本的链接组放进 nav 元素即可。例如，在页脚中通常会有一组链接，包括服务条款、首页和版权声明等，这时使用 footer 元素是最恰当的。一个页面可以拥有多个 nav 元素，作为页面整体或不同部分的导航。在 nav 元素中，一般以 ul 列表的形式来具体放置该组链接元素。

#### □ aside 元素

aside 元素用来表示当前页面或文章的附属信息部分，它可以包含与当前页面或主要内容相关的引用、侧边栏、广告、导航条，以及其他有别于主要内容的部分。

aside 元素主要有以下两种使用方法。

- 被包含在 article 元素中作为主要内容的附属信息部分，其中的内容可以是与当前文章有关的参考资料和名词解释等。
- 在 article 元素之外的元素中使用，作为页面或站点全局的附属信息部分。最典型的形式是侧边栏，其中的内容可以是友情链接、博客中其他文章列表和广告单元等。

#### □ hgroup 元素

hgroup 元素是将标题及其子标题进行分组的元素。hgroup 元素通常对 h1 ~ h6 元素进行分组，如将一个内容区块的标题及其子标题划为一组。

通常，如果一个 article 元素或 section 元素只有一个主标题，是不需要 hgroup 元素的。但是，如果该元素内有一个主标题，主标题下又有一个或多个副标题，就需要使用 hgroup 元素了，代码如下所示。

```
<article>
  <!-- 2 个标题元素被统一放置在一个 hgroup 元素内 -->
  <hgroup>
    <h1> 主标题 </h1>
    <h2> 副标题 </h2>
  </hgroup>
  正文
</article>
```

#### □ header 元素

header 元素是一种具有引导和导航作用的结构元素，通常用来放置整个页面或页面内的一个 article 元素或 section 元素的标题，也可以包含其他内容，例如数据表格、搜索表单或相关的 logo 图片。

#### □ footer 元素

footer 元素可以作为其上层父级内容区块或一个根区块的脚注。footer 元素通常包括其相关区块的脚注信息，如作者、相关阅读链接以及版权信息等。

### 3. HTML 5 中的大纲

在 Word 文档中，一份文档的大纲如下：

- 1. HTML 5 的基础知识
  - 1.1 HTML 5 概述
    - (第一章中第一节的正文)
  - 1.1.1 HTML5 是什么?
    - (第一章中第一节第一小节的正文)
  - 1.1.2 HTML5 中的新增 API
    - (第一章中第一节第二小节的正文)
  - 1.2 HTML5 快速入门
    - (第一章中第二节的正文)
  - 1.2.1 HTML 与 XHTML
    - (第一章中第二节第一小节的正文)

HTML 5 网页文档中的大纲也基本如此，只不过使用不同的结构元素来进行表达而已。换句话说，在 HTML 5 中，使用各种结构元素所描述出来的整个网页的层次结构，就是该网页的大纲。因此，在组织这份大纲的时候，不能使用 div 元素，因为 div 元素只能被当做容器，用在需要对网页中某个局部使用整体样式时。

有许多工具可以对 HTML 5 的网页文档进行分析，然后将该文档中的大纲以可视化的形式展现出来。前面提到的“<http://gsnedders.html5.org/outliner/>”网站中就有一个文档大纲分析工具，利用这分析工具对代码清单 1-4 所示文档进行分析，分析结果如图 1-1 所示。

代码清单 1-4 大纲分析工具测试用代码

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title>大纲分析</title>
</head>
<section>
  <h1>HTML5 的基础知识</h1>
  <section>
    <h2>HTML5 概述</h2>
    (正文)
    <section>
      <h3>HTML5 是什么?</h3>
```

```

        (正文)
    </section>
    <section>
        <h3>HTML5 中的新增 API</h3>
        (正文)
    </section>
</section>
<section>
    <h2>HTML5 快速入门 </h2>
    (正文)
    <section>
        <h3>HTML 与 XHTML</h3>
        (正文)
    </section>
</section>
</section>

```

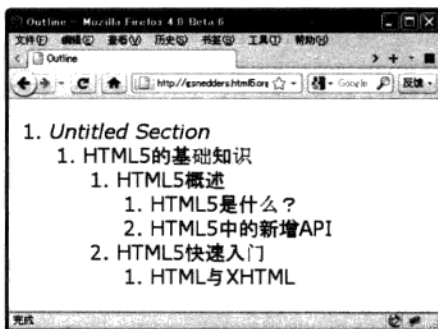


图 1-1 在线大纲分析工具的分析结果

图 1-1 中出现“1.Untitled Section”，是由于该网页文档中第一个元素即为 section 元素，缺乏整个网页标题元素。加入标题元素（h1），将代码清单 1-4 中的代码修改为如代码清单 1-5 所示，利用在线大纲分析工具分析出来的大纲如图 1-2 所示。

代码清单 1-5 添加了 header 元素后的大纲分析工具测试用代码

```

<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title> 大纲分析 </title>
</head>
<h1>HTML5 的基础知识</h1>
<section>
    <h2>HTML5 概述</h2>
    (正文)
    <section>
        <h3>HTML5 是什么？</h3>
        (正文)
    
```



```

</section>
<section>
  <h3>HTML5 中的新增 API</h3>
  (正文)
</section>
</section>
<section>
  <h2>HTML5 快速入门 </h2>
  (正文)
  <section>
    <h3>HTML 与 XHTML</h3>
    (正文)
  </section>
</section>

```

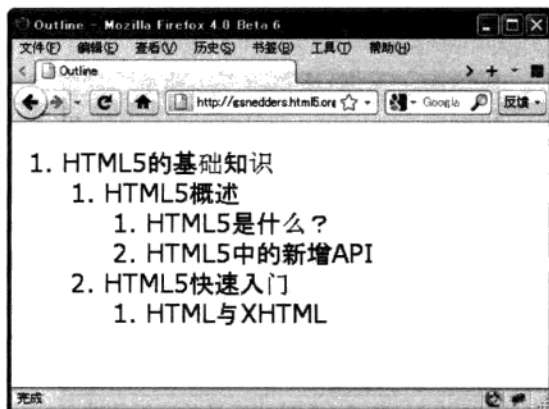


图 1-2 加入网页标题后的页面大纲

看到这里，你也许会问，如果只加一个 h1 元素就可以分析出标题来，那么，还需要 header 元素干什么？答案是 h1 元素一般是用来显示文字标题的。事实上，在要定义为网页标题的整个内容中，往往并不只是显示文字，其中包括了大量的导航条、企业的 logo 图片和广告 Flash 等，这些内容都可以放在 header 元素中，作为整个网页标题的内容，而标题文字为 h1 元素中的文字，在大纲中显示该标题文字。但是，这里要说明的是，header 元素本身的作用不是生成大纲，大纲是依靠 header 元素中的 h1 ~ h6 元素来生成的，如果使用代码清单 1-6 中的代码，则生成的大纲如图 1-3 所示。

代码清单 1-6 在企业网站中使用图片来显示企业名称

```

<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title>企业网站</title>
</head>
<header>

```

```


</header>
<section>
    <h2> 企业描述 </h2>
    (正文)
</section>

```

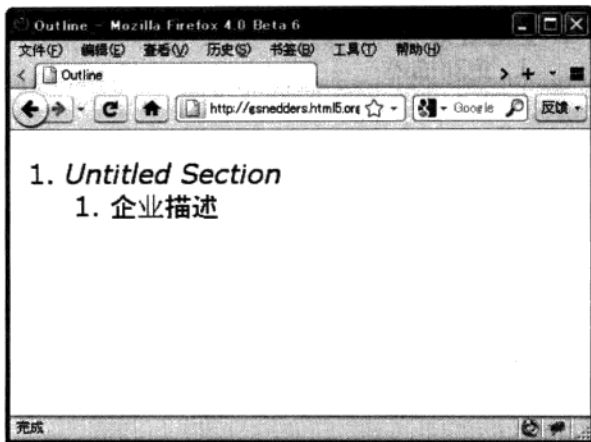


图 1-3 header 元素本身并不用于生成大纲

这里有这样一个问题，在很多企业网站（或其他网站）中，企业的标题并不是以文字来显示的，而是为了视觉效果，放在图片中显示的。难道这种情况就不能生成大纲了吗？笔者认为，这里有个小技巧，在 header 元素中，使用如下代码，既可以用图片来显示企业名称，又可以生成大纲。

```

<header>
    <h1></h1>
</header>

```

在 header 元素中使用这段代码后，生成的大纲如图 1-4 所示。

与 header 元素相同，footer 元素中如果没有标题元素（h1 ~ h6 元素）也不用于生成大纲。

在代码清单 1-4 或代码清单 1-6 的底部追加如下代码，生成的大纲将不会有任何变化。

```

<footer>
    版权所有：陆凌牛
</footer>

```

另外，对于 nav 元素和 aside 元素来说，如果不在元素内部加入标题元素（h1 ~ h6 元素），生成大纲时会在该元素所在位置处添加一个“Untitled Section”的说明文字，但是根据 HTML 5 的开发文档记载，nav 元素的作用是存放一组链接组，aside 元素的作用是表示当前页面或文章的附属信息部分，允许不在这两个元素中添加标题，也就是说，大纲中存在对于

这两个元素的内容为“Untitled Section”的说明文字是合理的。



图 1-4 在 header 元素中使用图片来生成大纲

在代码清单 1-4 的底部添加如下代码，生成的大纲如图 1-5 所示。

```
<nav>
  <ul>
    <li><a href="#">链接测试 1</a></li>
    <li><a href="#">链接测试 2</a></li>
  </ul>
</nav>
<aside>
  侧边栏中的内容
</aside>
```

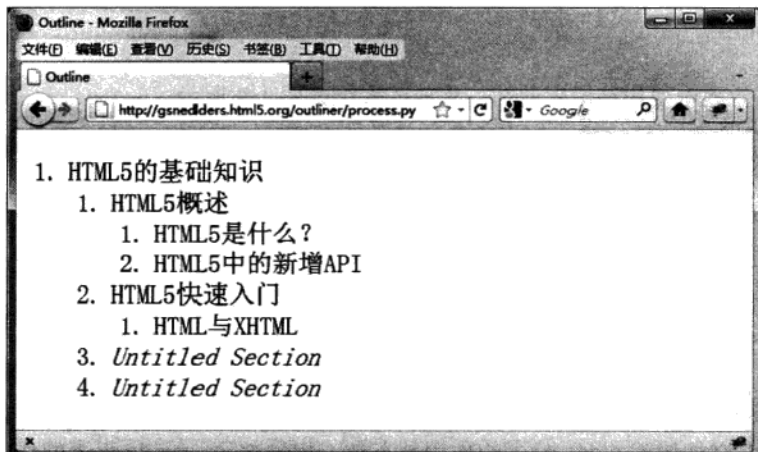


图 1-5 在文档的底部添加 nav 元素与 aside 元素所生成的大纲

另外, 在 HTML 5 中, body 元素、blockquote 元素、fieldset 元素、td 元素、details 元素及 figure 元素被称为节根 (sectioning roots) 元素。这些元素的共同特征是拥有自己独立的大纲, 并且这些元素内的 section 元素、article 元素、标题元素 (h1 ~ h6 元素)、nav 元素以及 aside 元素等, 只用在生成其父元素的大纲时, 而不用在生成父元素的上层祖先元素的大纲时。

在代码清单 1-7 中, blockquote 元素内部有一个 h1 元素, 正是因为这个 h1 元素是位于 blockquote 元素内部的, 所以在针对 blockquote 元素的父元素 body 生成页面大纲时, 该 h1 元素并没有显示在大纲中, 如图 1-6 所示。

代码清单 1-7 针对 body 元素生成大纲时节根元素中的子元素不起作用

```
<!DOCTYPE html>
<meta charset="UTF-8">
<body>
<h1> 网页标题 </h1>
<blockquote>
<h1> 节根元素内部标题 </h1>
</blockquote>
</body>
```

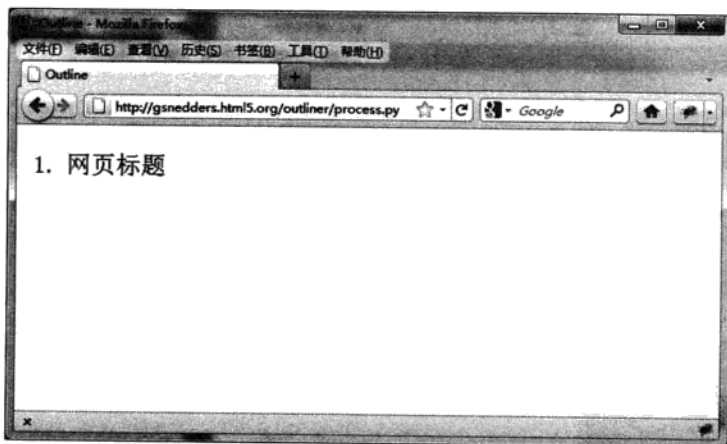


图 1-6 针对 body 元素生成大纲时节根元素中的子元素不起作用

掌握了 HTML 5 的文档结构、结构元素以及大纲的生成原则之后, 就可以学习如何使用这些基础知识来搭建一个语义清晰、结构分明的 HTML 5 网站了。下面先介绍一个利用 HTML 5 中的各种结构元素搭建出来的博客网站。

### 1.1.2 博客首页的实现

#### 1. 页面显示效果

接下来先来看一下这个案例中的博客首页在浏览器中的显示效果, 如图 1-7 所示。

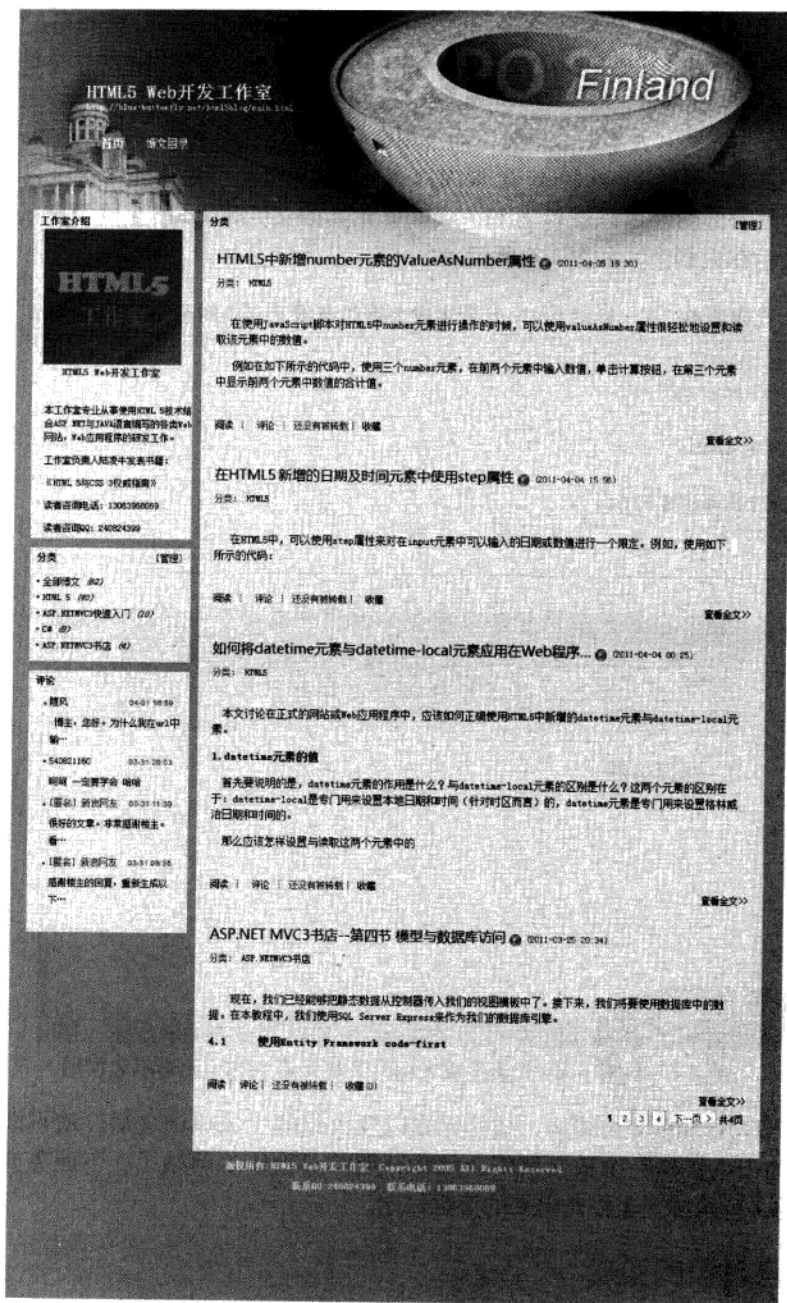


图 1-7 博客首页

该页面主要分为四个部分：第一部分为网页标题部分，显示该博客网站的网站标题与网

站导航链接；第二部分为网页侧边栏，显示博主自我介绍内容、博客中文章的所有分类链接以及网友对博客中文章的最新评论；第三部分为由博客中文章摘要组成的文章列表，即该博客首页中的主要内容；第四部分为页面底部的版权信息显示部分。

该页面的主体结构如图1-8所示。

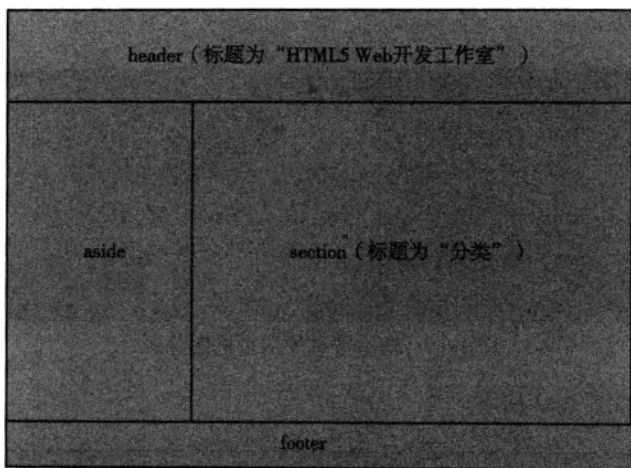


图1-8 博客首页结构图

该页面生成的大纲如图1-9所示。

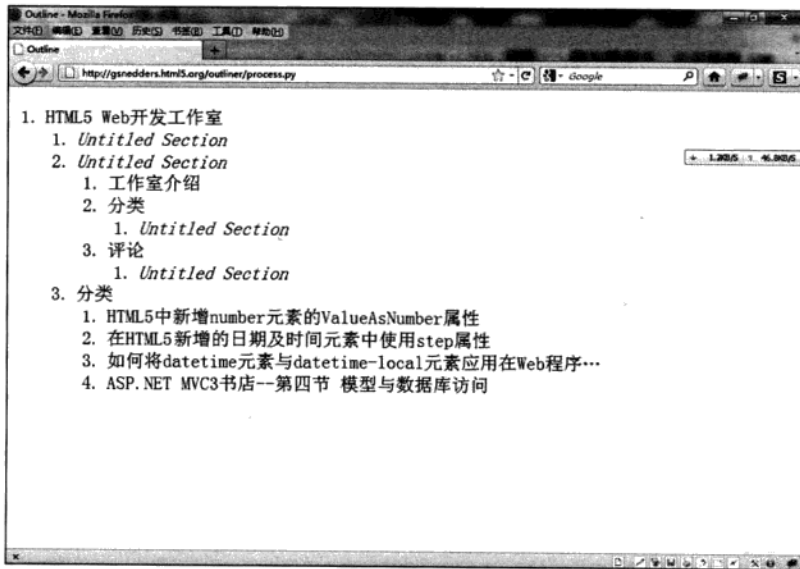


图1-9 博客首页生成的大纲

接下来详细介绍如何在该网页的各个组成部分中使用各种结构元素来搭建整体网页结

构，以及在大纲中为什么会有几个说明为“Untitled Section”的节，出现这样的节是否是正常的、合理的。

## 2. 构建网页标题

首先来看一下该网页中用来显示网站标题与网站导航的网页标题部分，该部分的页面显示效果如图 1-10 所示。



图 1-10 博客首页的网页标题部分

该部分的结构如图 1-11 所示。

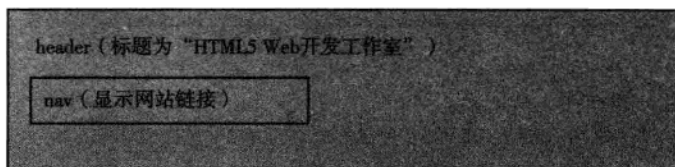


图 1-11 博客首页的网页标题部分的结构图

前面介绍过，header 元素是一种具有引导和导航作用的结构元素，通常用来放置整个页面或页面内的一个 article 元素或 section 元素的标题。在博客首页中，一般将博客的标题与整个网站的导航链接作为整体网页的标题放置在 header 元素中。

另外，在 header 元素内部使用了一个 nav 元素。如前所述，nav 元素是一个可以作为页面导航的链接组，其中的导航元素链接到其他页面或当前页面的其他部分。这里将整个网站的导航链接放在该 nav 元素中。

该部分的结构代码如下所示。

```
<header>
<h1>HTML5 Web 开发工作室 </h1>
<nav>
...
</nav>
</header>
```

在该网页中，由这部分结构代码生成的大纲如图 1-12 所示。

## 1. HTML5 Web开发工作室

### 1. *Untitled Section*

图 1-12 博客首页的网页标题部分生成的大纲

由于该网页使用了一个 header 元素来显示网页标题，并且在 header 元素的内部使用了 h1 元素，元素中的文字为“HTML5 Web 开发工作室”，因此整个大纲的标题为“1.HTML5 Web 开发工作室”。在 header 元素内部，使用 nav 元素来显示整个网站的导航链接，并且没有给 nav 元素添加标题（在 HTML 5 中，并不强求对 nav 元素添加标题），所以这个没有标题的 nav 元素在大纲中生成标题为“1.Untitled Section”的节。

最后，通过代码清单 1-8 来了解这个 header 元素的代码（包括整个网页的开头部分）。

代码清单 1-8 网页标题部分的代码

```
<!DOCTYPE html>
<meta charset="utf-8">
<title>HTML5 Web 开发工作室 </title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<meta content="" name="keywords">
<meta content="" name="description">
<link href="main.css" type="text/css" rel="stylesheet"/>
<div id="blog">
  <header id="bloghead">
    <div id="blogTitle">
      <h1 id="blogname">HTML5 Web 开发工作室 </h1>
      <div id="bloglink">
        http://blue-butterfly.net/html5blog/main.html
      </div>
    </div>
    <nav id="blognav">
      <ul id="blognavInfo">
        <li>
          <a href="http://blue-butterfly.net/html5blog/main.html"
            id="on">
            首页
          </a>
        </li>
        <li>
          <a href="articlelist_1792358825_0_1.html">博文目录 </a>
        </li>
      </ul>
    </nav>
  </header>
```

在这段代码中，在整个 body 元素（HTML 5 中可以将 body 元素省略不写）内部放置了一个作为容器的 div 元素，以显示该网页的背景图，样式代码如下所示。



```
div#blog {  
    background-position: 50% 0%;  
    background-image: url(images/blogb.jpg);  
    width: 100%;  
    background-repeat: no-repeat;  
}
```

使用 ul 列表元素来显示网站导航链接，并在样式代码中使用 list-style 属性控制列表编号不被显示，样式代码如下所示。

```
ul{  
    list-style:none;  
}
```

3. 构建侧边栏

接下来看一下该网页中用来显示博主介绍、博客中所有文章分类以及网友评论的侧边栏部分。

该部分在浏览器中的显示结果如图 1-13 所示。

该部分的结构如图 1-14 所示。

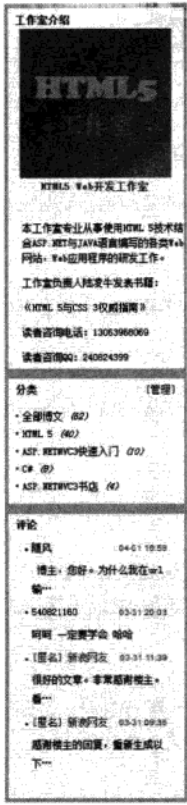


图 1-13 博客首页的侧边栏部分

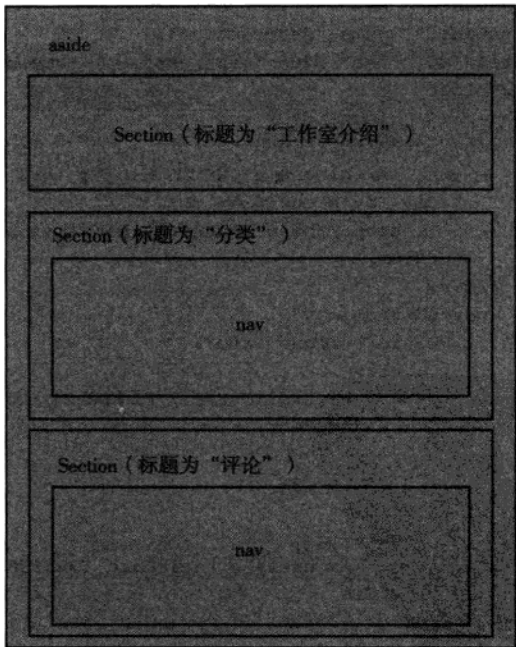


图 1-14 博客首页的侧边栏部分的结构图

前面介绍过，在 HTML 5 中，aside 元素专门用来表示当前页面或文章的附属信息部分，它可以包含与当前页面或主要内容相关的引用、侧边栏、广告、导航条，以及有别于主要内

容的部分。在博客首页中，可以将博主介绍、博主的联系信息、博客文章的分类、最近访问的网友信息、网友对博客文章的评论、相关文章的链接、其他网站的友情链接等很多与网站相关的但不能包含在当前网页的主体内容中的其他附属内容放置在 `aside` 元素中。在本例中，将博主介绍、博客文章分类及其链接（单击链接后主画面跳转到该分类的文章目录显示画面）、网友评论及其链接（单击链接后跳转到被评论的文章显示画面）放在了侧边栏中。

用来显示侧边栏的 `aside` 元素的内部结构代码如下所示。

```
<aside>
  <section>
    <header>
      <h1> 工作室介绍 </h1>
    </header>
    ...
  </section>
  <section>
    <header>
      <h1> 分类 </h1>
    </header>
    <nav>
      ...
    </nav>
  </section>
  <section>
    <header>
      <h1> 评论 </h1>
    </header>
    <nav>
      ...
    </nav>
  </section>
</aside>
```

在该网页中，侧边栏部分生成的大纲如图 1-15 所示。

- 2. *Untitled Section*
    - 1. 工作室介绍
    - 2. 分类
      - 1. *Untitled Section*
    - 3. 评论
      - 1. *Untitled Section*

图 1-15 侧边栏部分生成的大纲

HTML 5 会根据一个 `aside`（侧边栏）元素在大纲中生成与之对应的一个节。在本例中，由于没有对侧边栏添加标题（在 HTML 5 中，不强求对侧边栏添加标题），而且侧边栏位于整体网页结构中的第二部分（第一部分为网页标题），因此在大纲中生成标题为“2. *Untitled Section*”的节。在 `aside` 元素中，因为使用了三个 `section` 元素，分别显示博主介绍、博客文

章分类、网友评论这三个栏目的内容，并且三个 section 元素内部都有一个 header 元素，在 header 元素内部都使用了 h1 标题元素，标题文字分别为“工作室介绍”、“分类”、“评论”，所以在大纲中的侧边栏一节（“2.Untitled Section”）内部分别生成标题为“1. 工作室介绍”、“2. 分类”与“3. 评论”的节。在博客文章分类栏目与网友评论栏目中使用了两个 nav 元素来分别显示博客文章的分类及其链接与网友的评论及其链接，所以在大纲中，根据两个 nav 元素分别生成两个标题为“1.Untitled Section”的节。

在博主介绍栏目中，使用 figure 元素来显示博主头像。在 HTML 5 中，figure 元素用来表示网页上一块独立的内容，将其从网页上移除后不会对网页上的其他内容产生任何影响。figure 元素所表示的内容可以是图片、统计图或代码示例。

figcaption 元素表示 figure 元素的标题，它从属于 figure 元素，必须书写在 figure 元素内部，可以书写在 figure 元素内的其他从属元素的前面或后面。一个 figure 元素内最多允许放置一个 figcaption 元素，但允许放置多个其他元素。

本例中 figure 元素中的代码如下所示。

```
<figure>
  
  <figcaption>HTML5 Web 开发工作室 </figcaption>
</figure>
```

可以在样式代码中分别指定 figure 元素与 figcaption 元素的样式，代码如下所示。

```
figure{
  margin-left:15px;
  margin-top:0px;
}
figcaption{
  color:gray;
  font-weight:bold;
  padding-left:25px;
}
```

在网友评论栏目中，使用 time 元素与 pubdate 属性来显示每篇评论的发布时间。在 HTML 5 中，time 元素代表 24 小时中的某个时刻或某个日期，time 元素的 putdate 属性代表了评论的发布日期和时间（关于 figure 元素与 time 元素的详细介绍，可以参阅笔者所著《HTML 5 与 CSS3 权威指南》一书），代码类似下面所示。

```
<time datetime="2011-04-01T16:59" pubdate>04-01 16:59</time>
```

可以对 time 元素使用样式，代码如下所示。

```
time{
  color:#637160;
  font-size:12px;
}
```

侧边栏部分的完整代码如代码清单 1-9 所示。

代码清单 1-9 博客首页的侧边栏部分的代码

```
<div id="blogbody">
  <div id="column_1">
    <aside>
      <section id="conn1">
        <header id="connHead1">
          <h1> 工作室介绍 </h1>
        </header>
        <div id="connBody1">
          <div>
            <figure>
              HTML5 Web 开发工作室 </figcaption>
            </figure>
          </div>
          <div id="divSpecial">
            <p> 本工作室专业从事使用 HTML 5 技术结合 ASP.NET 与 JAVA 语言
            编写的各类 Web 网站, Web 应用程序的研发工作。</p>
            <p> 工作室负责人陆凌牛发表书籍: </p>
            <p>
              <a target="_blank"
                href="http://product.dangdang.com/product.aspx?
                product_id=21047278&ref=search-1-pub">
                《HTML 5 与 CSS 3 权威指南》
              </a>
            </p>
            <p> 读者咨询电话: 13063968069</p>
            <p> 读者咨询 QQ: 240824399</p>
          </div>
        </div>
        <div id="connFoot1"></div>
      </section>
      <section id="conn2">
        <header id="connHead2">
          <h1> 分类 </h1>
          <span id="edit1">
            <a href="javascript:;"
            onclick="window.CateDialog.show();return false;">
              [<cite> 管理 </cite>]
            </a>
          </span>
        </header>
        <div id="connBody2">
          <nav id="classList">
```

[illegible]

```

        &nbsp;博主，您好。为什么我在 url 中输...
    </a>
</div>
</div>
</li>
<li id="commentsCell_linedot2">
    <div id="commentsH2">
        <span id="commentsName_txtc_dot2">
            <a href="#" target="_blank"
                title="540821160">540821160</a>
        </span>
        <time datetime="2011-03-31T20:03"
            pubdate>03-31 20:03</time>
    </div>
    <div id="commentsContants2">
        <div id="commentsContantsTxt2">
            <a href="#" target="_blank">
                呵呵 &nbsp;一定要学会 &nbsp;哈哈
            </a>
        </div>
    </div>
</li>
<li id="commentsCell_linedot3">
    <div id="commentsH3">
        <span id="commentsName_txtc_dot3">
            [ 匿名 ] 新浪网友
        </span>
        <time datetime="2011-03-31T11:39"
            pubdate>03-31 11:39</time>
    </div>
    <div id="commentsContants3">
        <div id="commentsContantsTxt3">
            <a href="#" target="_blank">
                很好的文章。非常感谢楼主。看...
            </a>
        </div>
    </div>
</li>
<li id="commentsCell_linedot4">
    <div id="commentsH4">
        <span id="commentsName_txtc_dot4">
            [ 匿名 ] 新浪网友
        </span>
        <time datetime="2011-03-31T09:35"
            pubdate>03-31 09:35</time>
    </div>
    <div id="commentsContants4">
        <div id="commentsContantsTxt4">

```

```

        <a href="#" target="_blank">
            感谢楼主的回复，重新生成以下...
        </a>
    </div>
</div>
</li>
</ul>
</nav>
</div>
<div id="connFoot3"></div>
</section>
</aside>
</div>

```

---

在这段代码中，第一行的 id 为“blogbody”的 div 是一个容器元素，使用它将该网页的当中一块（包括左边的侧边栏区域与右边的文章摘要列表区域）与网页顶部的标题区域及网页底部的脚注区域（显示版权信息的 footer 元素）区分开来。该 div 的样式代码如下所示。

```

div#blogbody{
    margin: 0px;
}

```

在 id 为“blogbody”的 div 元素的内部，又使用了一个 id 为“column\_1”的 div 元素，用来将左边的侧边栏部分与右边的网页主体部分进行区分。注意，这段代码中最后一行的“</div>”元素结束标记仅表示 id 为“column\_1”的 div 元素的结束，并不表示 id 为“blogbody”的 div 元素的结束。

id 为“column\_1”的 div 元素的样式代码如下所示。

```

[id^=column_]{
    display:inline;
    float:left;
    overflow:hidden;
}
div#column_1{
    margin-left:20px;
    width:210px;
}

```

#### 4. 构建网页主体内容

接下来看一下这个博客首页中的主体显示内容部分，即文章摘要列表显示部分。该部分在浏览器中的显示结果如图 1-16 所示。

该部分的整体内容被放置在一个 section 元素中，该 section 元素的内部结构如图 1-17 所示。

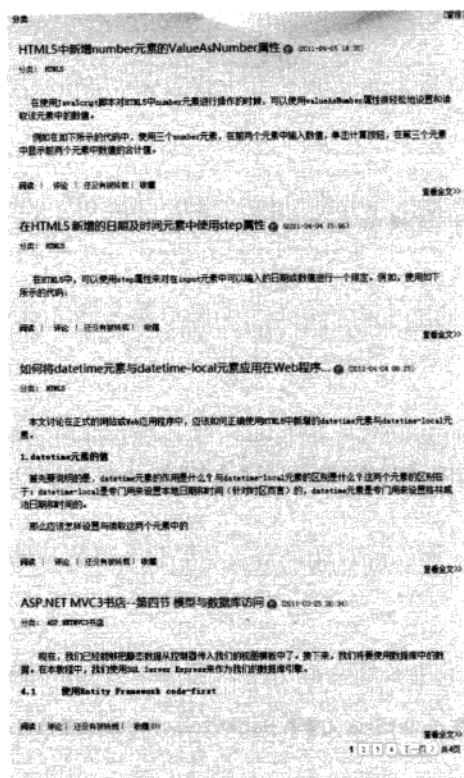


图 1-16 博客首页的主体内容部分

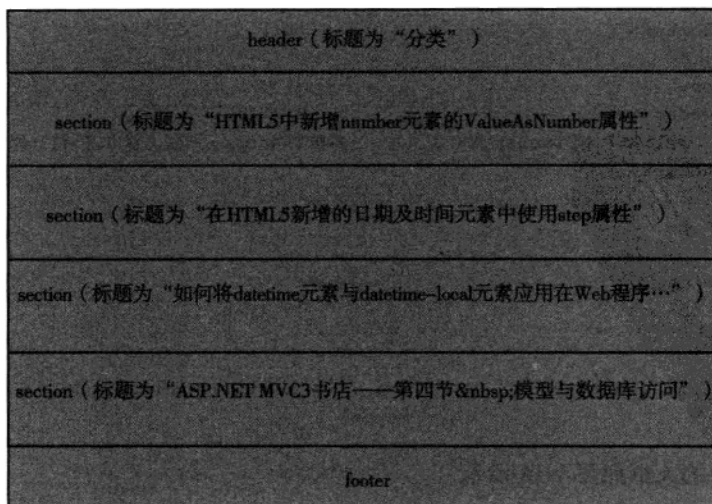


图 1-17 显示博客首页的主体内容部分的 section 元素的内部结构图



该 section 元素内部的结构代码如下所示。

```
<section>
  <header>
    <h1> 分类 </h1>
  </header>
</section>
<section>
  <header>
    <h1>HTML5 中新增 number 元素的 ValueAsNumber 属性 </h1>
  </header>
  ...
  <footer>
    ...
  </footer>
</section>
<section>
  <header>
    <h1> 在 HTML5 新增的日期及时间元素中使用 step 属性 </h1>
  </header>
  ...
  <footer>
    ...
  </footer>
</section>
<section>
  <header>
    <h1> 如何将 datetime 元素与 datetime-local 元素应用在 Web 程序...</h1>
  </header>
  ...
  <footer>
    ...
  </footer>
</section>
<section>
  <header>
    <h1>ASP.NET&nbsp;&MVC3 书店 -- 第四节 &nbsp;& 模型与数据库访问 </h1>
  </header>
  ...
  <footer>
    ...
  </footer>
</section>
<footer>
  ...
</footer>
</section>
```

该部分生成的大纲如图 1-18 所示。

3. 分类

  1. HTML5 中新增 number 元素的 ValueAsNumber 属性
  2. 在 HTML5 新增的日期及时间元素中使用 step 属性
  3. 如何将 datetime 元素与 datetime-local 元素应用在 Web 程序...
  4. ASP.NET MVC3 书店——第四节 模型与数据库访问

图 1-18 根据显示博客首页的主体内容的 section 元素生成的大纲

在这个 section 元素内部,使用了一个 header 元素、四个 section 元素、一个 footer 元素,其中 header 元素的标题为“分类”,所以在大纲中生成标题为“3. 分类”的节。这个 section 元素内部的四个 section 元素中又各自有一个 header 元素,其中都存放了一个显示标题的 h1 元素,标题分别显示为“HTML5 中新增 number 元素的 ValueAsNumber 属性”、“在 HTML5 新增的日期及时间元素中使用 step 属性”、“如何将 datetime 元素与 datetime-local 元素应用在 Web 程序...”和“ASP.NET MVC3 书店——第四节 &nbsp; 模型与数据库访问”,所以在大纲中分别生成标题为“1. HTML5 中新增 number 元素的 ValueAsNumber 属性”、“2. 在 HTML5 新增的日期及时间元素中使用 step 属性”、“3. 如何将 datetime 元素与 datetime-local 元素应用在 Web 程序...”和“4. ASP.NET MVC3 书店——第四节 &nbsp; 模型与数据库访问”的四个节。另外,四个 section 元素中又各自有一个 footer 元素,存放每篇文章的阅读链接(单击链接后打开该文章)、评论链接(单击链接后打开该文章并跳转到评论部分)、被转载次数与收藏链接(单击链接后收藏该文章)。在显示网页主体内容部分的 section 元素的结尾处又使用了一个 footer 元素,显示对文章摘要列表进行分页。由于 footer 元素中没有标题元素(h1~h6 元素)用于生成大纲,所以在大纲中没有根据这些 footer 元素生成任何节。

显示博客首页的主体内容部分的 section 元素的内部代码如代码清单 1-10 所示。

代码清单 1-10 显示博客首页的主体内容的 section 元素的内部代码

```
<div id="column_2">
  <section id="conn4">
    <header id="connHead4">
      <h1> 分类 </h1>
      <span id="edit2">
        <a href="javascript:;"
          onclick="window.CateDialog.show();return false;"
          [<cite> 管理 </cite>]>
        </a>
      </span>
    </header>
    <div id="connBody4">
      <div id="bloglist">
        <section>
          <header>
            <div id="blog_title_h1">
              <h1 id="blog_title1">
                <a href="#" target="_blank">
                  HTML5 中新增 number 元素的 ValueAsNumber 属性
                </a>
              </h1>
            </div>
          </header>
        </section>
      </div>
    </div>
  </section>
</div>
```

[illegible]



datetime-local 元素的区别是什么？这两个元素的区别在于：  
datetime-local 是专门用来设置本地日期和时间（针对时区而言）的，  
datetime 元素是专门用来设置格林威治日期和时间的。

    那么应该怎样设置与读取这两个元素中的

```
</div>
<footer id="tagMore3">
    <div id="tag_txtc3">
        <a href="#" target="_blank">阅读</a>&nbsp;&nbsp;&nbsp;; |&nbsp;&nbsp;&nbsp;
        <a target="_blank" href="#">评论</a>
        &nbsp;&nbsp;&nbsp;; |&nbsp;&nbsp;&nbsp;还没有被转载！&nbsp;&nbsp;&nbsp;
        <a href="javascript:;" onclick="return false;">收藏
    </a>
    </div>
<div id="more3">
    <span id="smore3">
        <a href="#" target="_blank">查看全文</a>&gt;&gt;
    </span>
</div>
</footer>
</section>
<section>
    <header>
        <div id="blog_title_h4">
            <h1 id="blog_title4">
                <a href="#" target="_blank">ASP.NET&nbsp;&MVC3 书店 -- 第四节 &nbsp;&模型与数据库访问
            </a>
        </h1>
        
        <time datetime="2011-03-25T20:34" pubdate>(2011-03-25 20:34)</time>
    </div>
    <div id="articleTag4">
        <span id="txtb4">分类:</span>
        <a target="_blank" href="#">ASP.NETMVC3 书店</a>
    </div>
</header>
<div id="content4">
    <p>
        &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&~
        现在，我们已经能够把静态数据从控制器传入我们的视图模板中了。接下来，我们将要使用数据库中的数据。在本教程中，我们使用 SQL Server Express 来作为我们的数据库引擎。
    </p>
    <p>
        <strong>
            <span>
```



除了这个充当容器的 `div` 元素之外，代码中的其他地方也使用了比较多的 `div` 元素来作为对某个局部区域使用样式的容器元素，这里不再一一列出样式代码。注意，该代码中最后一行的“`</div>`”这个结束标记，用于结束代码清单 1-9 中的 `id` 为“`blogbody`”的 `div` 元素，表示网页中央部分（与网页顶部的网页标题部分及网页底部的版权信息显示部分进行区分）已经结束。

## 5. 构建版权信息

最后来看一下该页面中位于网页底部的版权信息显示部分，该部分在浏览器中的显示效果如图 1-19 所示。



图 1-19 版权信息显示部分

该部分被放置于一个 footer 元素中，因为没有使用标题，所以也没有被显示在大纲中。该部分代码如代码清单 1-11 所示。

代码清单 1-11 版权信息显示部分的代码

[illegible]

该部分代码的最后一个“</div>”元素结束标记表示代码清单 1-8 中 id 为“blog”的 div 元素已经结束。

### 1.1.3 文章显示页面的实现

接下来看一下在该博客网站中打开某篇文章时显示的文章显示页面，该页面在浏览器中显示的效果如图 1-20 所示。

该页面与博客首页有些相似，主要分为四个部分，第一部分为网页标题部分，显示该博客网站的网站标题、网站链接与网站导航。第二部分为网页侧边栏，显示博主自我介绍内容、博客文章的所有分类链接以及网友对博客中文章的最新评论。这两部分与首页中的网页标题部分及侧边栏部分完全相同。第三部分为查看文章的内容及网友评论部分，也是该页面的主要内容。第四部分为页面底部显示的版权信息内容部分。

该页面的主体结构也与博客首页的主体结构大致相同，只是在博客首页中使用 section 元素来显示文章摘要列表，而在文章显示页面中，使用 section 元素来显示文章内容与网友

的评论内容。该页面的主体结构图也与博客首页中的主体结构图相似，如图1-21所示。

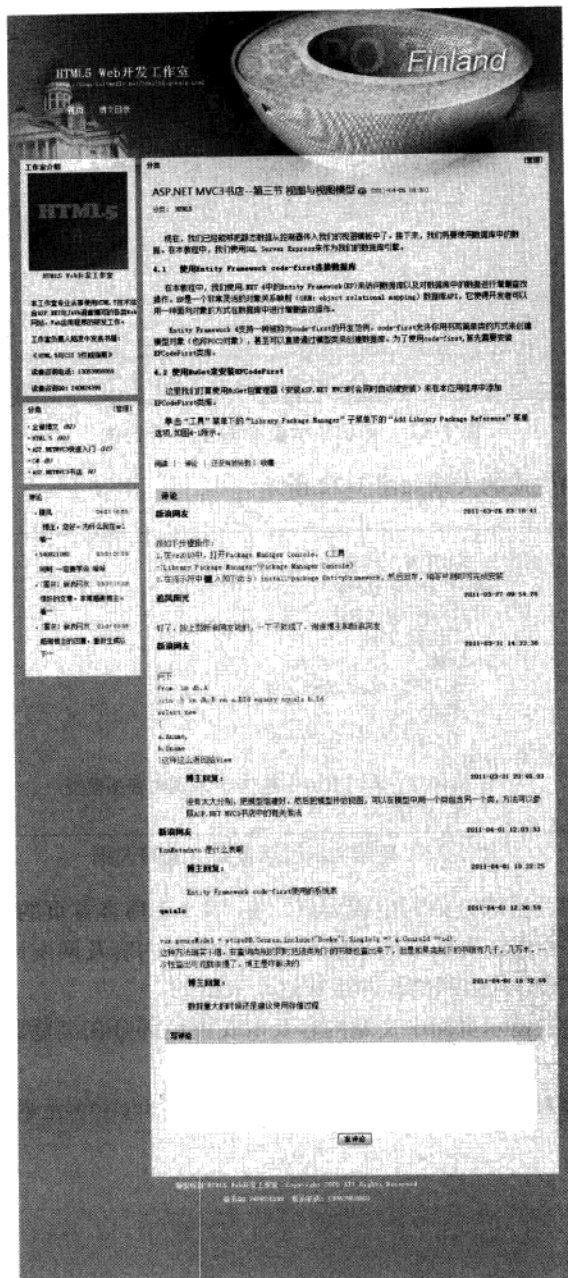


图 1-20 文章显示页面



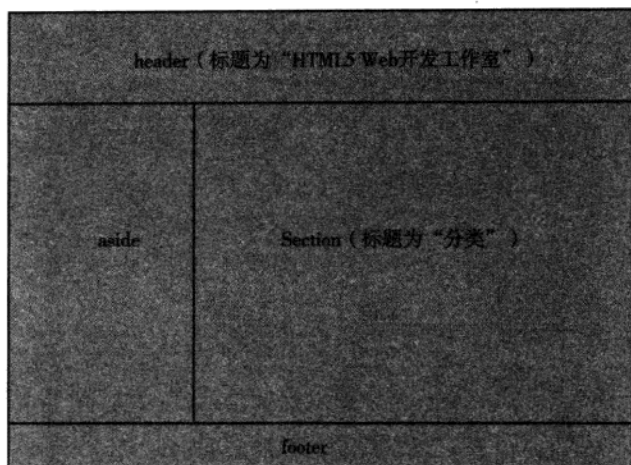


图 1-21 文章显示页面的主体结构图

根据文章显示页面生成的大纲如图 1-22 所示。

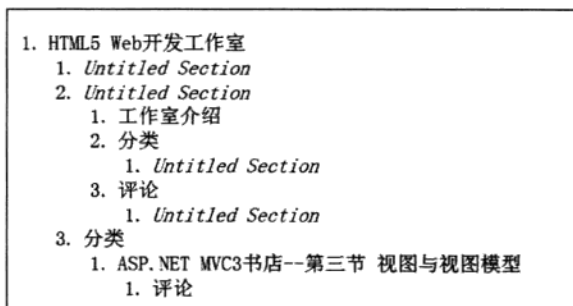


图 1-22 根据文章显示页面生成的大纲

网页标题部分与侧边栏部分如何生成大纲，在“1.1.2 博客首页的实现”中已经介绍过，这里不再赘述。接下来介绍该页面中的主体部分，即文章内容及网友评论部分中是如何使用各种结构元素来搭建该部分的组织结构并生成这个大纲的。

接下来重点介绍文章显示页面中文章内容及网友评论部分在浏览器中的显示效果，如图 1-23 所示。

该部分的整体内容被放置在一个 section 元素中，该 section 元素的内部结构如图 1-24 所示。

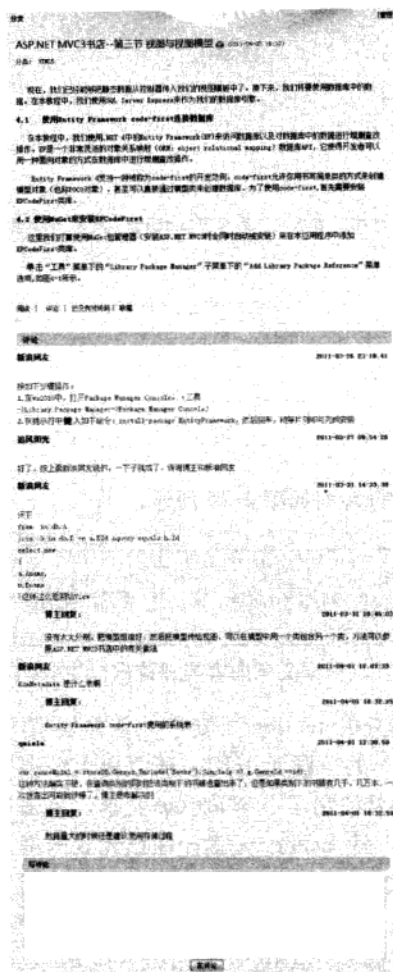


图 1-23 文章内容及网友评论部分

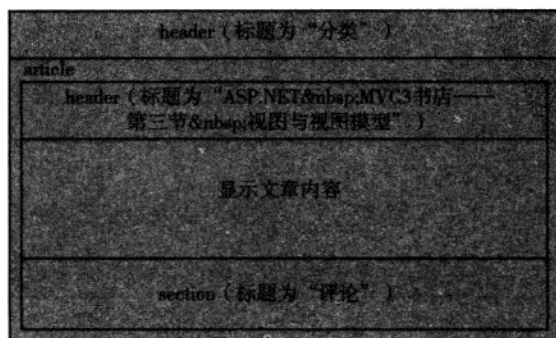


图 1-24 显示文章内容及网友评论的 section 元素的内部结构图

该 section 元素内部的结构代码如下所示。

```
<section>
  <header>
    <h1> 分类 </h1>
  </header>
  <article>
    <header>
      <h1>ASP.NET MVC3 书店 -- 第三节 &nbsp;视图与视图模型 </h1>
    </header>
    ...
    <section>
      <h4> 评论 </h4>
      ...
    </section>
  </article>
</section>
```

在该网页中, 该部分生成的大纲如图 1-25 所示。

3. 分类 1. ASP.NET MVC3书店—第三节 视图与视图模型 1. 评论
---

图 1-25 根据文章内容与评论部分生成的大纲

在显示文章内容与评论部分的 section 元素中, 首先使用了一个 header 元素, 在该元素内部使用了一个文字为“分类”的标题元素 h1, 所以在大纲中生成一个标题为“3. 分类”的节。在 header 元素后面, 紧接着使用了一个 article 元素, 用来显示文章内容与网友评论。在这个 article 元素内部, 使用了一个 header 元素, 在该元素内部, 又使用了一个文字为“ASP.NET MVC3 书店——第三节 视图与视图模型”的标题元素 h1, 所以在大纲中生成一个标题为“1. ASP.NET MVC3 书店——第三节 视图与视图模型”的节。在 article 元素内部, 在 header 元素后面, 显示了标题为“ASP.NET MVC3 书店——第三节 视图与视图模型”的文章的全部内容, 在文章叙述完毕之后, 使用了一个 section 元素, 在这个元素内部, 又使用了一个 header 元素。在这个 header 元素之中, 又使用了一个文字为“评论”的标题元素 h4, 所以在大纲中生成一个标题为“1. 评论”的节。

另外, 在标题为“评论”的 section 元素中, 使用了一个 Iframe 内嵌网页, 在该网页中使用了个表单, 在该表单中放置了一个用来写评论内容的 textarea 元素, 以及一个用来提交评论内容的“发评论”按钮。

最后, 通过代码清单 1-12 来了解显示文章内容与网友评论的 section 元素中的全部内容。

代码清单 1-12 用来显示文章内容与网友评论的 section 元素的完整代码

```
<div id="column_2">
  <section id="conn4">
```

[illegible]



```

        <a href="#" target="_blank">追风阳光 </a>
    </span>
    <span id="revert_Time2">
        <time datetime="2011-03-27T09:54:26"
            pubdate>
            2011-03-27 09:54:26
        </time>&nbsp;
    </span>
</p>
<div id="revert_Inner_txtb2">
    <br>好了，按上面新浪网友说的，一下子就成了。谢谢
    博主和新浪网友
</div>
</div>
</li>
<li id="linedot3">
    <div id="revert_Cont3">
        <p>
            <span id="revert_Tit3">新浪网友 </span>
            <span id="revert_Time3">
                <time datetime="2011-03-31T14:33:38"
                    pubdate>
                    2011-03-31 14:33:38
                </time>&nbsp;
            </span>
        </p>
        <div id="revert_Inner_txtb3">
            <br>问下 <br>from&nbsp;&nbsp;&nbsp;in&nbsp;&nbsp;db.A
            <br>join&nbsp;&nbsp;&nbsp;b&nbsp;&nbsp;in&nbsp;&nbsp;
            db.B&nbsp;&nbsp;on&nbsp;&nbsp;a.Bid&nbsp;&nbsp;equerry
            &nbsp;&nbsp;equals&nbsp;&nbsp;b.Id
            <br>select&nbsp;&nbsp;new&nbsp;&nbsp;<br>{
            <br>a.Aname,<br>b.Bname<br>} 这样这么返回给
            View
        </div>
        <div id="nlinedot1">
            <p>
                <span id="revert_Tit4">博主回复: </span>
                <span id="revert_Time4">
            </p>
            <p id="myReInfol">
                没有太大分别，把模型组建好，然后把模型传给视图，
                可以在模型中用一个类包含另一个类，方法可以参照
                ASP.NET&nbsp;MVC3 书店中的有关做法
            </p>
        </div>
    </div>
</li>
<li id="linedot4">
    <div id="revert_Cont4">

```

```

<p>
  <span id="revert_Tit5"> 新浪网友 </span>
  <span id="revert_Time5">
    <time datetime="2011-04-01T12:03:33"
      pubdate>
      2011-04-01 12:03:33
    </time>&nbsp;
  </span>
</p>
<div id="revert_Inner_txtb4">
  EdmMetadata&nbsp;是什么表啊
</div>
<div id="nlinedot2">
  <p>
    <span id="revert_Tit6"> 博主回复: </span>
    <span id="revert_Time6">
      <time datetime="2011-04-01T12:03:33"
        pubdate>
        2011-04-01 12:03:33
      </time>&nbsp;
    </span>
  </p>
  <p id="myReInfo2">
    Entity&nbsp;Framework&nbsp;code-first
    使用的系统表
  </p>
</div>
</div>
</li>
<li id="linedot5">
  <div id="revert_Cont5">
    <p>
      <span id="revert_Tit7">
        <a href="#" target="_blank">qmialo</a>
      </span>
      <span id="revert_Time7">
        <time datetime="2011-04-01T12:30:59"
          pubdate>
          2011-04-01 12:30:59
        </time>&nbsp;
      </span>
    </p>
    <div id="revert_Inner_txtb5">
      <br>var&nbsp;genreModel&nbsp;= &nbsp;
      storeDB.Genres.Include("Books").Single(g
      &nbsp; &gt;&nbsp;g.GenreId&nbsp;==id);
      <br>这种方法确实不错，在查询类别的同时把该类别下
      的书籍也查出来了，但是如果类别下的书籍有几千，几万
      本，一次性查出可能就很慢了，博主是咋解决的
    </div>
  </div>
  <div id="nlinedot3">
    <p>
      <span id="revert_Tit8"> 博主回复: </span>
      <span id="revert_Time8">
        <time

```

```

        datetime="2011-04-01T18:32:59"
        pubdate>
        2011-04-01 18:32:59
    </time>
</span>
</p>
<p id="myReInfo3">数据量大的时候还是建议使用
存储过程</p>
</div>
</div>
</li>
</ul>
</div>
<div id="writeComm">
    <iframe src="writeComm.jsp" width="90%" height="300">
    </iframe>
</div>
</section>
</article>
</div>
</section>
</div>

```

## 1.2 案例 2：用 HTML 5 中的结构元素构建一个企业网站

本节通过一个使用 HTML 5 中各种结构元素构建企业网站的案例来讲述如何使用 HTML 5 中的各种新增元素，搭建出一个语义清晰、结构分明的 HTML 5 版的企业网站。本节所介绍的网站是根据江苏省常州市蓝博纺织机械有限公司正在运行的 HTML 4 版的网站改编的，而非该企业正在运行的网站。

本案例主要讲述该网站的两个页面，一个页面为该网站的首页，即本节的主要内容，另一个为该网站的联系方式页面，作为补充内容，该网站中其他页面结构均与这两个页面类似，故不做介绍。

### 1.2.1 首页的实现

#### 1. 页面显示效果

首先来看一下本节要介绍的网站的首页在浏览器中的显示效果，如图 1-26 所示。

该页面主要分为四个部分，第一部分为网页标题部分，显示该企业网站的网站标题，网站导航链接。第二部分为网页侧边栏，显示该企业的产品分类与联系方式。第三部分为企业网站首页中所展示的主要内容，第四部分为页面底部的企业版权信息与联系方式显示。

该页面的整体结构图如图 1-27 所示。





图 1-26 网站的首页

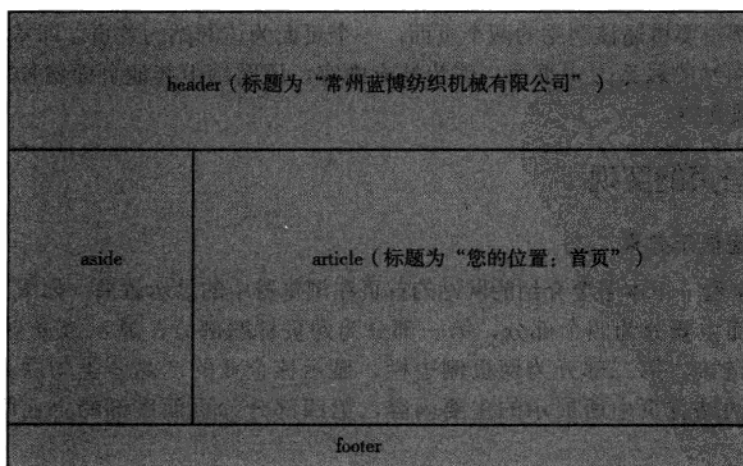


图 1-27 网站首页的整体结构图

该页面生成的大纲如图 1-28 所示。

1. 常州蓝博纺织机械有限公司
  1. *Untitled Section*
  2. *Untitled Section*
    1. *Untitled Section*
3. 您的位置：首页
  1. 公司动态
  2. 公司简介
  3. 推荐产品

图 1-28 网站首页的大纲

接下来具体介绍在该页面的各个部分中是如何使用 HTML 5 的各种结构元素来搭建这个首页的组织结构并生成这个大纲的。

## 2. 构建网页标题

首先来看一下该首页的网页标题部分。该部分在浏览器中的显示效果如图 1-29 所示。

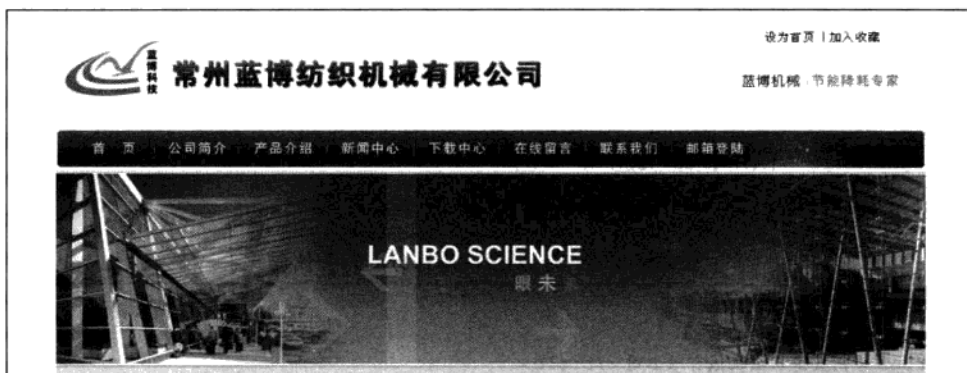


图 1-29 网站首页的网页标题部分

该部分被放置在一个 header 元素中。在企业网站中，通常将企业名称、企业 logo 图片、整个网站的导航链接，以及一些广告图片、广告 Flash 等放置在 header 元素中，作为网页标题部分。

网页标题部分的结构图如图 1-30 所示。

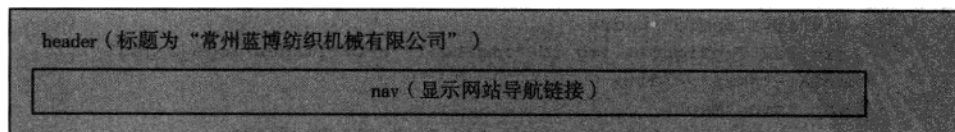


图 1-30 网页标题部分的结构图

在这个网页标题部分中，使用了一个 nav 元素来显示整个网站的导航链接。该部分的结构代码如下所示。

```
<header>
<h1>

</h1>
<nav>
...
</nav>
...
</header>
```

在该网站首页中，由于使用图片来显示企业名称，因此通过将图片放在 h1 元素中并将图片元素 img 的 alt 属性设置为企业名称的办法在大纲中将企业名称作为整个网页的标题。

在该网页中，这段结构代码生成的大纲如图 1-31 所示。

## 1. 常州蓝博纺织机械有限公司

### 1. *Untitled Section*

图 1-31 网页标题部分生成的大纲

由于在 header 元素中使用了 nav 元素来显示网站导航链接，并且没有给这个 nav 元素添加标题，因此生成“1.Untitled Section”一节。

网页标题部分的完整代码如代码清单 1-13 所示。

代码清单 1-13 网页标题部分的完整代码

---

```
<!DOCTYPE html>
<meta charset=utf-8 />
<title> 常州蓝博纺织机械有限公司 </title>
<link href="main.css" type="text/css" rel="stylesheet"/>
<header id="webTitle">
  <div id="divTop1">
    <div id="TopLeft">
      <h1>
        
      </h1>
    </div>
    <div id="TopMid"></div>
    <div id="TopRight"></div>
    <map name="Map">
      <area shape="rect" coords="32,5,93,21" href="#"
        alt=" 常州蓝博纺织机械有限公司 ">
      <area shape="rect" coords="103,3,167,22" href="#"
```

```

        alt="常州蓝博纺织机械有限公司" >
    </map>
</div>
<nav>
    <ul id="topNavUrl">
        <li>
        </li><li>
            <a href="index.asp"></a>
        </li><li>
        </li><li>
            <a href="about.asp">
                
            </a>
        </li><li>
        </li><li>
            <a href="product.asp"></a>
        </li><li>
        </li><li>
            <a href="news.asp"></a>
        </li><li>
        </li><li>
            <a href="download.asp"></a>
        </li><li>
        </li><li>
            <a href="gbook.asp"></a>
        </li><li>
        </li><li>
            <a href="contact.asp"></a>
        </li><li>
        </li><li>
            <a href="http://mail.czlbkj.cn"></a>
        </li><li></li>
    </ul>
</nav>
<div id="sectionTop2">
    <div id="Bottom1"></div>
    <div id="FlashDiv">
        <object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
            codebase="http://download.macromedia.com/pub/shockwave/
            cabs/flash/swflash.cab#version=6,0,29,0"
            type="application/x-shockwave-flash"
            width="900" height="200" >
            <param name="movie" value="images/index.swf">

```

```

        <param name="quality" value="high">
        <param name=wmode value=transparent>
        <embed src="images/index.swf"
              quality="high"
              pluginspage="http://www.macromedia.com/go/
              getflashplayer"
              type="application/x-shockwave-flash"
              width="900"
              height="200"/>

      </object>
    </div>
    <div id="Bottom2"></div>
  </div>
</header>

```

在这段代码中，在 nav 元素内部，使用了 ul 列表来显示网站导航链接，为了避免在 li 列表项目元素的背景中使用的图片与图片之间存在裂痕，必须将前一个 li 元素的结束标记 </li> 与后一个 li 元素的开始标记 <li> 写在同一行中，书写成 “</li><li>” 的形式。另外，在样式代码中，需要使用如下的代码，使 li 列表项目元素的项目编号不显示，并且并排显示。

```

<ul#topNavUrl li{
  list-style::none;
  display: inline-block;
}

```

### 3. 构建侧边栏

接下来介绍网站首页的侧边栏部分，该部分在浏览器中的显示效果如图 1-32 所示。

该侧边栏部分的结构图如图 1-33 所示。

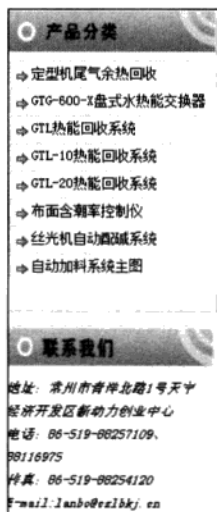


图 1-32 网站首页的侧边栏部分

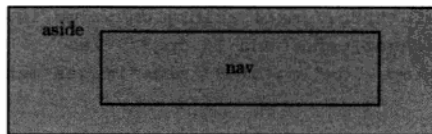


图 1-33 侧边栏部分的结构图

在企业网站中，一般可以将所有产品分类以及产品分类下每个产品的页面链接作为一个链接组放在 nav 元素中，然后将该 nav 元素放在侧边栏中，也可以将企业的联系方式和友情链接等用户浏览整个网站时始终可以看见的重要信息放在侧边栏中。

这个示例网站首页的侧边栏分为两个部分，第一部分是用来显示所有产品分类及其产品链接的 nav 元素，第二部分为该企业的联系信息。由于直接使用 ul 列表元素直接显示企业的

联系信息，因此在结构中第二部分没有体现出来。

侧边栏部分的结构代码如下所示。

```
<aside>
  <nav>
    ...
  </nav>
  ...
</aside>
```

由于没有对显示侧边栏的 `aside` 元素及其内部显示产品分类链接组的 `nav` 元素使用标题（在HTML 5中，不强求对 `aside` 元素以及 `nav` 元素使用标题），因此生成的大纲如图1-34所示。

```
2. Untitled Section
1. Untitled Section
```

图1-34 根据侧边栏及其内部的 `nav` 元素生成的大纲

该侧边栏元素的内部代码如代码清单1-14所示。

代码清单1-14 示例网站中侧边栏元素的内部代码

```
<aside>
  <nav>
    
    <ul>
      <li><a href="show.asp?cid=29">定型机尾气余热回收</a></li>
      <li><a href="show.asp?cid=27">GTG-600-X 盘式水热能交换器</a></li>
      <li><a href="show.asp?cid=25">GTL 热能回收系统</a></li>
      <li><a href="show.asp?cid=18">GTL-10 热能回收系统</a></li>
      <li><a href="show.asp?cid=19">GTL-20 热能回收系统</a></li>
      <li><a href="show.asp?cid=17">布面含潮率控制仪</a></li>
      <li><a href="show.asp?cid=16">丝光机自动配碱系统</a></li>
      <li><a href="show.asp?cid=20">自动加料系统主图</a></li>
    </ul>
    
  </nav>
  <div>
    <ul>
      <li></li>
      <li><address>地址：常州市青洋北路1号天宁经济开发区新动力创业中心</address></li>
      <li><address>电话：86-519-88257109、<br/>88116975</address></li>
      <li><address>传真：86-519-88254120</address></li>
      <li><address>E-mail:lanbo@czlbkj.cn</address></li>
      <li></li>
    </ul>
  </div>
</aside>
```

在这段代码中, 使用了 address 元素来显示企业的联系信息。在 HTML 5 中, address 元素是一种专门用来显示联系信息的元素, 可以使用它来显示网站链接、电子邮箱、真实地址和电话号码等各种联系信息。

#### 4. 构建主体内容

接下来看一下该网站首页的主要展示的内容部分, 该部分在浏览器中的显示效果如图 1-35 所示。



图 1-35 网站首页的主要展示内容部分

该部分被放置在一个 article 元素中, 该 article 元素的结构图如图 1-36 所示。

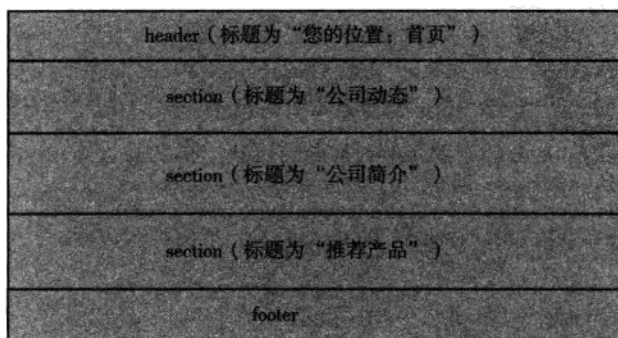


图 1-36 显示首页中主要展示内容的 article 元素的结构图

该 article 元素内部的结构代码如下所示。

```
<article>
  <header><h1>您的位置：首页</h1></header>
  <section>
    <h1>
      
    </h1>
    ...
  </section>
  <section>
    <h1>
      
    </h1>
    ...
  </section>
  <section>
    <h1>
      
    </h1>
    ...
  </section>
</article>
```

在该网站首页中，由这段结构代码生成的大纲如图 1-37 所示。

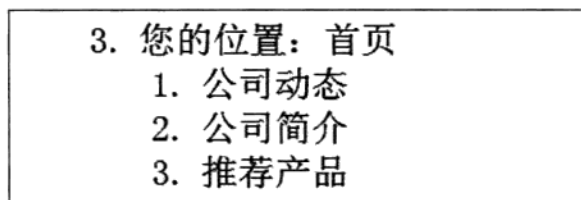


图 1-37 首页主要展示内容部分生成的大纲

在这个 article 元素内部，首先使用了一个 header 元素，header 元素内部又放置了一个标题元素 h1，标题文字为“您的位置：首页”，该部分在大纲中生成的标题为“3. 您的位置：首页”。在 article 元素内部，还放置了三个 section 元素，其中分别放置了标题“公司动态”、“公司简介”与“推荐产品”，所以在大纲中分别生成标题为“1. 公司动态”、“2. 公司简介”与“3. 推荐产品”的三个节。

这个 article 元素的内部代码如代码清单 1-15 所示。

代码清单 1-15 显示首页中主要展示内容的 article 元素的内部代码

```
<article id="main" >
  <header><h1>您的位置：首页</h1></header>
  <section id="section1">
    <div id="topdiv1">
```



```

        <h1>
            
        </h1>
    </div>
    <div id="bottomdiv1">
        <div id="div1" >
            <ul>
                <li>
                    <ul>
                        <li id="li1_1"></li>
                        <li id="li1_2">
                            <a href="newsinfo.asp?id=45" class="path">
                                2011 年春节放假通知
                            </a>
                        </li>
                        <li id="li1_3">
                            <time datetime="2011-01-20" pubdate>
                                2011-1-20
                            </time>
                        </li>
                    </ul>
                </li>
                <li id="line1"></li>
                <li>
                    <ul>
                        <li id="li2_1"></li>
                        <li id="li2_2">
                            <a href="newsinfo.asp?id=44" class="path">
                                2010 纺织印染行业年会
                            </a>
                        </li>
                        <li id="li2_3">
                            <time datetime="2010-11-13" pubdate>
                                2010-11-13
                            </time>
                        </li>
                    </ul>
                </li>
                <li id="line2"></li>
                <li>
                    <ul>
                        <li id="li3_1"></li>
                        <li id="li3_2">
                            <a href="newsinfo.asp?id=42" class="path">
                                2010 中国柯桥国际纺织品博览会（秋季）
                            </a>
                        </li>
                        <li id="li3_3">
                            <time datetime="2010-11-02" pubdate>
                                2010-11-2

```

```

        </time>
    </li>
</ul>
</li>
<li id="line3"></li>
<li>
    <ul>
        <li id="li4_1"></li>
        <li id="li4_2">
            <a href="newsinfo.asp?id=41" class="path">
                声明</a>
        </li>
        <li id="li4_3">
            <time datetime="2010-08-17" pubdate>
                2010-8-17
            </time>
        </li>
    </ul>
</li>
<li id="line4"></li>
<li>
    <ul>
        <li id="li5_1"></li>
        <li id="li5_2">
            <a href="newsinfo.asp?id=40" class="path">
                高温下的激情
            </a>
        </li>
        <li id="li5_3">
            <time datetime="2010-08-17" pubdate>
                2010-8-17
            </time>
        </li>
    </ul>
</li>
<li id="line5"></li>
</ul>
</div>
<div id="div2">
    
</div>
</div>
</section>
<section id="section2">
    <div id="topdiv2">
        <h1>
            
        </h1>
    </div>
    <div id="bottomdiv2">

```

[illegible]

```

        <li>
            <a href="show.asp?cid=31">
                
                </a>
            </li>
        <li> 定型机尾气余热回收 </li>
    </ul>
</li>
<li>
    <ul>
        <li>
            <a href="show.asp?cid=31">
                
                </a>
            </li>
            <li>GTG-600-X 盘式水热能交换器 </li>
        </ul>
    </li>
</ul>
</div>
</section>
</article>

```

在 article 元素内部的第三个 section 元素中（“推荐产品”部分），使用了 ul 列表中嵌套 ul 列表的布局方式，外层 ul 列表中的 li 列表项目元素的排列方向为纵向排列，内层 ul 列表中的 li 列表项目元素的排列方向为横向排列。这些 ul 列表元素与 li 列表项目元素的样式代码如下所示。

```

li{
    font-weight: normal;
    font-size: 12px;
    color: #000000;
    line-height: 20px;
    font-family: 宋体;
    text-decoration: none;
    list-style:none;
}
article#main section [id^=bottomdiv]{
    display: -moz-box;
    display: -webkit-box;
    -moz-box-orient: horizontal;
    -webkit-box-orient:horizontal;
}
section#section3 #bottomdiv3 ul

```

```

{
    display: -moz-box;
    display: -webkit-box;
    -moz-box-orient: horizontal;
    -webkit-box-orient: horizontal;
    -moz-box-align: center;
    -webkit-box-align: center;
}
section#section3 #bottomdiv3 ul{
    width:620px;
    height:99px;
    list-style:none;
}
section#section3 #bottomdiv3 ul li{
    height:99px;
}
section#section3 #bottomdiv3 ul li ul{
    display: -moz-box;
    display: -webkit-box;
    -moz-box-orient: vertical;
    -webkit-box-orient: vertical;
    width:100%;
}
section#section3 #bottomdiv3 ul li ul li{
    align:center;
}

```

## 5. 构建版权信息

最后来看一下位于首页底部的版权信息显示部分，该部分在浏览器中的显示效果如图 1-38 所示。

版权所有：常州蓝博纺织机械有限公司 地址：常州市青洋北路1号天宁经济开发区新动力创业中心	苏ICP备06108115号 技术支持：常州蓝翔网络 电话：86-519-88257109、88116975 传真：86-519-88254120
---	--

图 1-38 版权信息显示部分

该部分被放置于一个 footer 元素中，因为没有使用标题，所以也没有被显示在大纲中。

本部分代码相对来说比较简单，只需使用适当的 div 元素，然后在其中放入版权信息文字即可，故不再赘述。

### 1.2.2 联系方式页面的实现

接下来介绍一下本网站中的联系方式页面，该页面在浏览器中的显示效果如图 1-39 所示。

该页面的整体结构图与网站首页的整体结构图大致相同，只是将显示网页主体内容的 article 元素中的内容换成了联系方式页面中的内容，如图 1-40 所示。



图 1-39 网站的联系方式页面

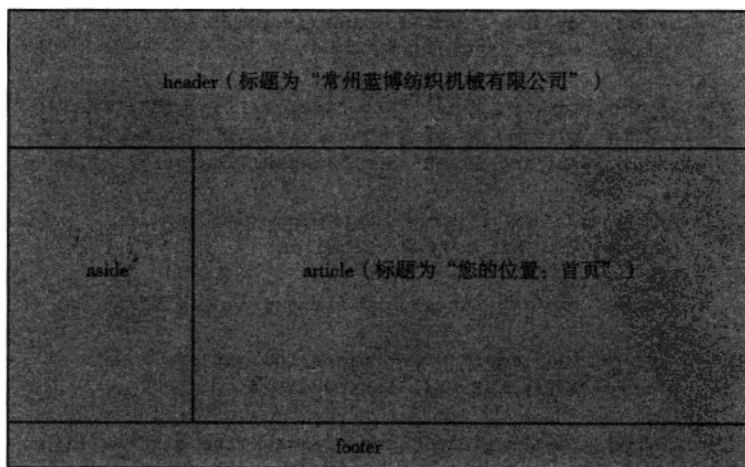


图 1-40 联系方式页面的整体结构图

该页面生成的大纲如图 1-41 所示。

1. 常州蓝博纺织机械有限公司
    1. *Untitled Section*
    2. *Untitled Section*
    1. *Untitled Section*
  3. 您的位置: 首页 >> 联系我们

图 1-41 联系方式页面生成的大纲

显示联系方式的 article 元素中的内部代码如代码清单 1-16 所示。

代码清单 1-16 联系方式页面中显示联系方式的 article 元素中的内部代码

---

```

<article id="main" >
<header><h1> 您的位置: 首页 &gt;&gt; 联系我们 </h1></header>
  <div id="top"></div>
  <div id="left"></div>
  <div id="right">
    <ul>
      <li></li>
      <li>常州蓝博纺织机械有限公司 </li>
      <li id="space"></li>
      <li>
        
        &nbsp;&nbsp;&nbsp; 传真: 86-519-88254120
      </li>
      <li></li>
      <li>
        
        &nbsp;&nbsp;&nbsp; 手机: 13186695633
      </li>
      <li></li>
      <li>&nbsp;&nbsp;&nbsp; 联系人: 冯先生 </li>
      <li></li>
      <li>
        
        &nbsp;&nbsp;&nbsp; 网址: http://www.czlbkj.cn
      </li>
      <li></li>
      <li>
        
        &nbsp;&nbsp;&nbsp; 邮件地址: E-mail:lanbo@czlbkj.cn
      </li>
      <li></li>
    </ul>
  </div>
</article>

```

---

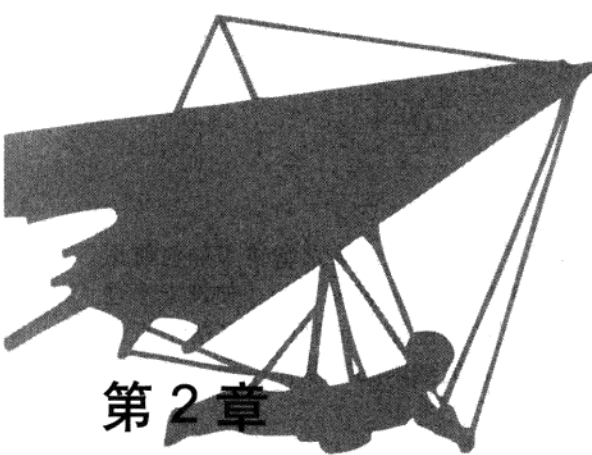
本网站中其他几个主要页面的整体结构均与这个联系方式页面的整体结构大致相同，只不过将显示网页主体内容的 `article` 元素中的内容换成每个页面中各自要展示的主体内容，故不再介绍。

## 1.3 本章小结

本章通过利用 HTML 5 制作的一个博客网站与一个企业网站，重点阐述了如何使用 HTML 5 的结构元素来制作语义清晰、结构分明的 Web 3.0 时代的网站，力求使读者通过本章的阅读，能够对 HTML 5 中的结构元素以及使用这些结构元素所制作出来的网页大纲有一个比较详细的了解，能够利用这些结构元素与网页大纲的知识制作出属于自己的 HTML 5 网站。

下一章将通过两个案例来阐述如何使用 HTML 5 中的结构元素与表单元素制作 Web 应用程序，如何对用户 HTML 5 中新增的表单元素中输入的内容进行验证，如何让服务器端能够正确获取用户在 HTML 5 中新增表单元素中输入的内容并将其保存在数据库中。





## 第 2 章

# 在 Web 表单中使用 HTML 5

### 本章内容

- ☐ 案例 3：用结构元素制作 Web 应用程序中的菜单
- ☐ 案例 4：综合运用 HTML5、jQuery 与 ASP.NET 构建 Web 应用程序
- ☐ 本章小结

本章通过两个案例来介绍如何在Web表单中使用HTML 5的新增结构元素和新增表单元素,以及怎样编写服务器端的脚本语言将用户在HTML 5新增表单元素中的输入内容保存到服务器端的数据库中。

本章首先介绍一个使用HTML 5、CSS 3样式和jQuery脚本制作的比较典型的Web应用中的菜单。接下来重点介绍一个使用HTML 5及其新增表单元素、CSS 3样式、jQuery脚本以及ASP.NET服务器端脚本开发的Web应用中的一个信息录入页面。该页面中有一个显示所有录入信息的一览表,用户录入一条信息后,该数据在数据库中保存完毕后立即显示在一览表中。该页面使用一个目前在Web网站或Web应用广泛使用的、使用jQuery脚本开发的jQuery验证器。该验证器在客户端对用户输入的各种数据(包括用户在HTML 5新增的各种表单元素中输入的各种数据)进行验证,验证通过后再将数据提交到服务器端进行保存。

## 2.1 案例3:用结构元素制作Web应用程序中的菜单

### 2.1.1 页面显示效果

在本案例中,我们将利用HTML 5的结构元素来制作一个Web应用程序中比较常见的菜单。该Web应用程序在浏览器中的总体效果如图2-1所示。

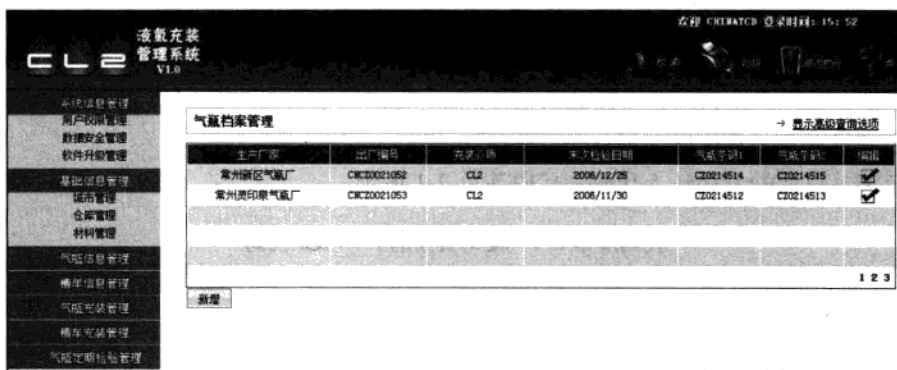


图 2-1 Web 应用程序的总体效果

这个Web应用程序的页面框架由三个子页面组成,分别为顶部的工具栏菜单页面、左边的主菜单页面与右边的应用程序主体页面。

这里只关注左边的应用程序主菜单页面,该页面打开时在浏览器中的显示效果如图2-2所示。

这个菜单有七个主菜单项,其中第三个菜单项到第七个菜单项没有子菜单,单击菜单项后应用程序右边的主体页面中会根据单击的菜单项显示不同的页面,而第一个菜单项和第二个菜单项均有子菜单项,



图 2-2 主菜单页面

单击主菜单项时页面中会显示该主菜单项下面的子菜单项，如图 2-3 和图 2-4 所示，再次单击该主菜单项时其子菜单项会被隐藏。

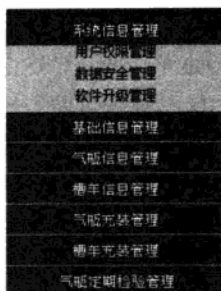


图 2-3 单击“系统信息管理”主菜单项时将显示其子菜单项

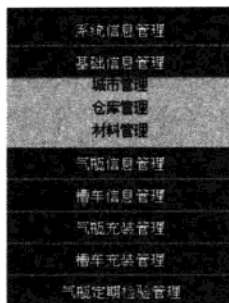


图 2-4 单击“基础信息管理”主菜单项时将显示其子菜单项

同样地，单击这些子菜单项后应用程序右边的主体页面中会根据单击的子菜单项显示不同的页面。

### 2.1.2 代码剖析

接下来先看一下这个菜单页面的 HTML 5 代码部分。在该页面中，由于各菜单项中均有链接元素 a，单击该菜单项后，右边的主体页面会由一个页面切换到另一个页面，所以可以将这些菜单视为一个链接组，并将其放置在 nav 元素中。在 nav 元素中，再由 ul 列表元素及其 li 列表项目元素来具体显示每一个菜单项。

这个菜单页面的 HTML 5 代码如代码清单 2-1 所示。

代码清单 2-1 菜单页面的 HTML 5 代码

```
<!DOCTYPE html>
<meta charset=utf-8 />
<title>HTML 5 版菜单</title>
<body onload="init()">
  <nav id="nav">
    <ul>
      <li id="first"></li>
      <li></li>
      <li>
        <nav id="submenu1">
          <ul>
            <li>
              <a href="add_user.html" target="mainFrame">
                用户权限管理
              </a>
            </li>
            <li>
```

```

        <a href="data_security.html" target="mainFrame">
            数据安全
        </a>
    </li>
    <li>
        <a href="soft_update.html" target="mainFrame">
            软件升级
        </a>
    </li>
</ul>
</nav>
</li>
<li></li>
<li>
    <nav id="submenu2">
        <ul>
            <li>
                <a href="add_city.html" target="mainFrame">
                    城市管理
                </a>
            </li>
            <li>
                <a href="add_store" target="mainFrame">
                    仓库管理
                </a>
            </li>
            <li>
                <a href="add_material.html" target="mainFrame">
                    材料管理
                </a>
            </li>
        </ul>
    </nav>
</li>
<li><a href="qpxx.html"></a></li>
<li><a href="ccgl.html"></a></li>
<li><a href="qpcz.html"></a></li>

<li><a href="cccz.html"></a></li>
<li><a href="qpjy.html"></a></li>
</ul>
</nav>
</body>
</html>

```

这个菜单页面中使用的样式代码如代码清单 2-2 所示。

代码清单 2-2 菜单页面所使用的样式代码

```

<style type="text/css">
body {
    margin:0px;

```

```

}
nav ul li{
    list-style:none;
    width:195px;
    line-height:10px;
}
nav ul li img{
    width:195px;
    height:28px;
    cursor:pointer;
}
nav ul li#first{
    height:6px;
    background:url(images/left_title_bg.gif);
}
nav ul li ul {
    display:inline;
    padding:0px;
}
nav ul li ul li{
    height:20px;
    background-color:#AEE4EE;
    font-size: 12px;
    color: #333333;
    text-align: center;
    width:195px;
}
nav ul li ul li a{
    font-size: 12px;
    color: #333333;
    text-decoration: none;
}
</style>

```

最后来看一下该菜单页面中使用的 JavaScript 脚本代码部分。该菜单页面使用的脚本代码比较简单，其功能为打开页面时将第一个主菜单项与第二个主菜单项下的子菜单项隐藏起来，单击第一个主菜单项或第二个主菜单项时将该主菜单项下的子菜单项显示出来，再次单击时将该主菜单项下的子菜单项隐藏起来，代码如代码清单 2-3 所示。

代码清单 2-3 菜单页面中使用到的脚本代码

```

<script>
function init()
{
    document.getElementById("submenu1").hidden=true;
    document.getElementById("submenu2").hidden=true;
}
function nav_fill1()
{

```

```

if (document.getElementById("submenu1").hidden===true)
    document.getElementById("submenu1").hidden=false;
else
    document.getElementById("submenu1").hidden=true;
}
function nav_fill2()
{
    if (document.getElementById("submenu2").hidden===true)
        document.getElementById("submenu2").hidden=false;
    else
        document.getElementById("submenu2").hidden=true;
}
</script>

```

## 2.2 案例 4：综合运用 HTML 5、jQuery 与 ASP.NET 构建 Web 应用程序

### 2.2.1 案例概述

在本案例中，我们将制作 Web 应用程序中的一个数据输入页面。该页面主要在企业内部的操作人员输入商业订单或企业订单时使用，故又叫订单输入页面。该页面分为上下两个部分，用户在页面上半部分的 Web 表单中输入一条数据，然后单击追加按钮，该数据将会在页面下半部分的一览表中即时显示出来，表示数据追加成功。

### 2.2.2 页面显示效果

首先来看一下该页面在浏览器中的显示效果，如图 2-5 所示。

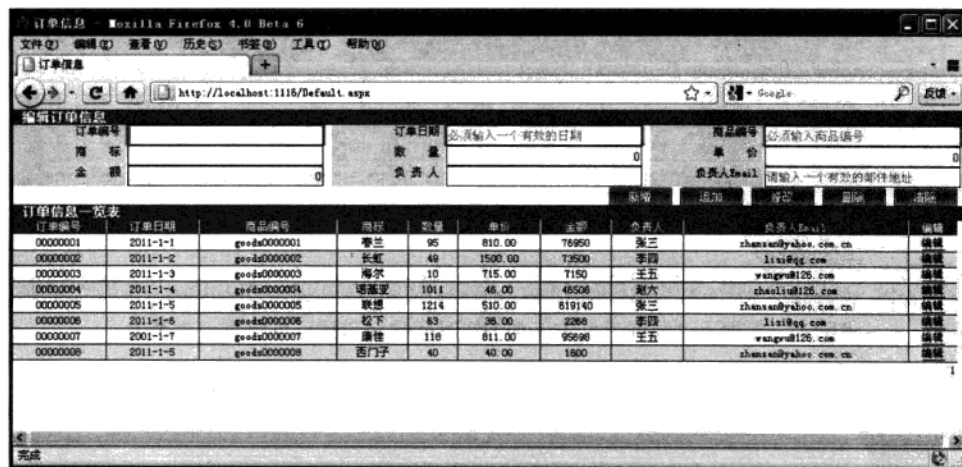


图 2-5 页面在浏览器中的显示效果

该页面具有自动验证功能,当用户在某个文本框中输入无效数据后单击追加按钮时,页面上自动显示验证错误提示信息,阻止表单向服务器端的提交,如图 2-6 所示。

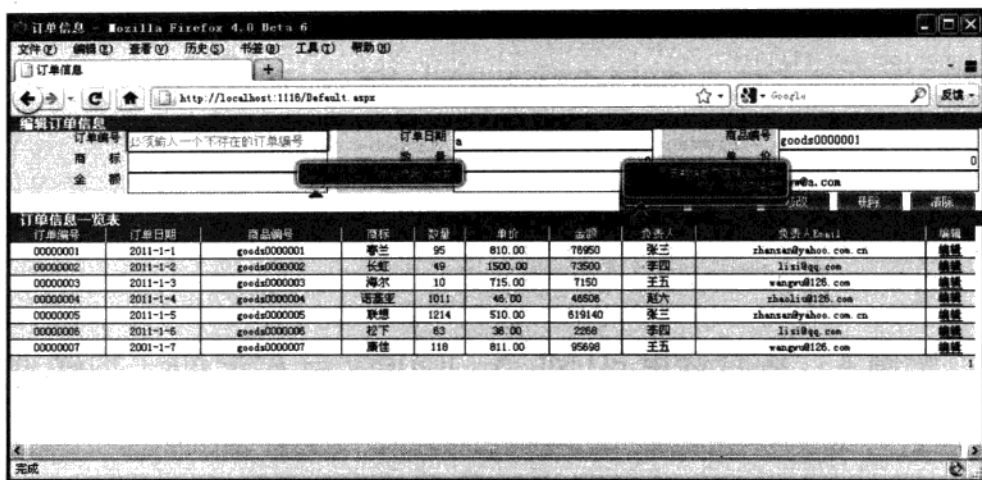


图 2-6 数据无效时显示验证错误信息

当用户修改该数据使其能够通过验证后,单击追加按钮,该数据将被保存到数据库中,然后即时显示在页面下端的数据一览表中,如图 2-7 所示。

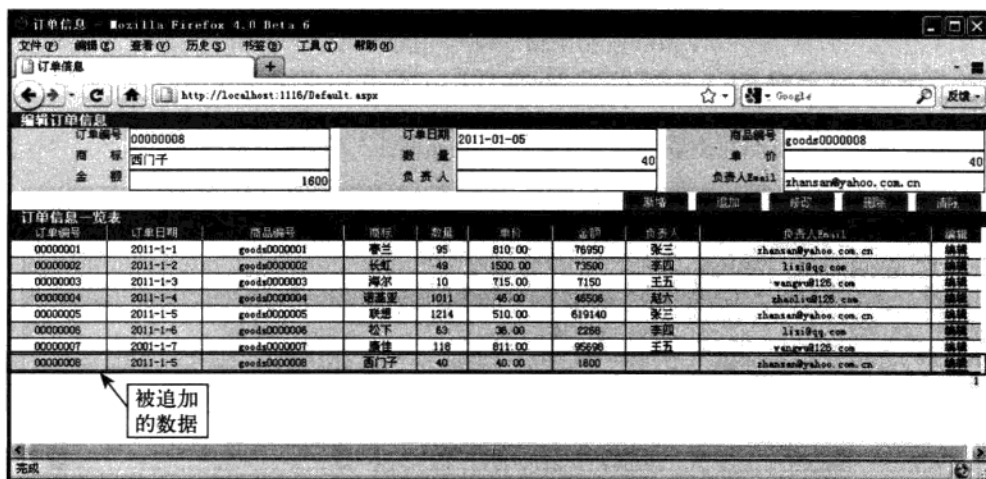


图 2-7 数据通过验证后单击追加按钮时即被显示到订单信息一览表中

该页面的结构比较简单,有上下两个 section 元素,如图 2-8 所示。

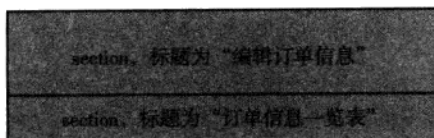


图 2-8 订单输入页面的结构图

这两个 section 元素内部开头处都没有使用 header 元素，而是直接使用标题元素 h5 来显示上下两个区域的标题文字。

### 2.2.3 案例知识点

本案例主要涉及以下几方面的知识点：

- HTML 5 中新增表单元素的作用以及使用方法
- jQuery 脚本代码的编写方法
- ASP.NET 服务器端脚本代码的编写方法

HTML 5 中新增表单元素的作用以及使用方法的有关知识，可以参照笔者所著的《HTML 5 与 CSS 3 权威指南》一书，此处不再赘述。

关于脚本代码以及 ASP.NET 服务器端脚本代码的编写方法，均不在本书所介绍的范围之中，故不做介绍。

这里对本案例所使用的一个 jQuery 验证器插件——jQuery-validation 插件进行详细介绍，使读者对这个插件有一个比较全面的认识。

jQuery-validation 插件可以从官方网站 (<http://cloud.github.com/downloads/posabsolute/jquery-Validation-Engine/formValidator.2.1.zip>) 上下载。当用户提交表单时该插件能够根据用户指定的验证规则对向表单中输入的元素内容进行验证，当验证没通过时给出具有悬浮效果的验证错误提示信息，如图 2-9 所示。

在这个页面中，由于使用了 jQuery 验证插件，因此当用户提交表单时，会对表单中所有元素进行验证。当验证没有通过时，给出具有悬浮效果的验证错误提示信息，如“2.2.2 页面显示效果”中的图 2-6 所示。

从官方网站上下载的压缩包中有很多文件，主要用到以下三个文件。

- (下载文件夹) \formValidator.2.1\js\jquery.validationEngine.js (插件主文件)
- (下载文件夹) \formValidator.2.1\js\languages\jquery.validationEngine-en.js (供用户添加验证规则的 js 文件，内置基本验证规则)
- (下载文件夹) \formValidator.2.1\css\validationEngine.jquery.css (验证插件所使用的样式表文件)

为了在页面中使用这个 jQuery 插件，我们需要执行以下几个步骤。

1) 在页面中加入如下所示的 jQuery 脚本代码。

```
$(function () {
```



```
$("#form1").validationEngine();
});
```

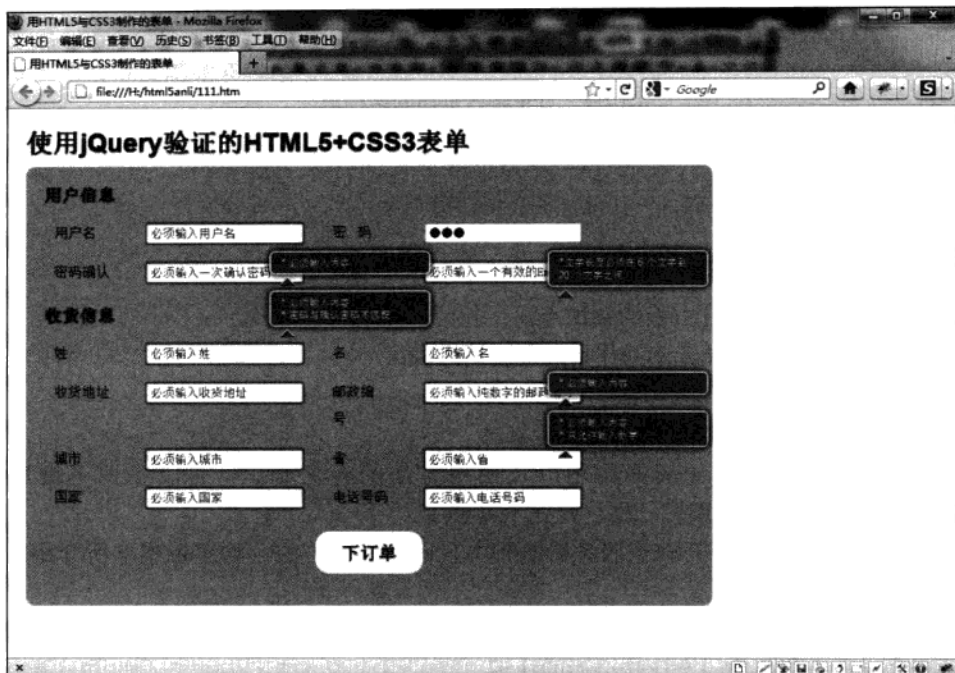


图 2-9 jQuery 验证插件的具有悬浮效果的验证错误提示信息

2) 在需要验证的元素中添加 class 属性。例如，为了验证页面中的“订单编号”文本框，我们为它添加了如下的 class 属性。

```
<input type="text" id="tbxCode" name="tbxCode" maxlength="8"
class="validate[required]" placeholder="必须输入一个不存在的订单编号"
autofocus required/>
```

本例使用了如下几个验证规则，各规则之间以“,”分隔。

- ❑ required: 必须在该元素中输入内容。
- ❑ optional: 该元素内容可以为空。之所以使用该验证规则是因为它之后往往具有其他验证规则，如果元素内容不为空则必须满足其他验证规则。如果不加入 optional 规则而直接书写验证规则表示元素内容必须满足验证规则，而且该元素内容不能为空。
- ❑ custom[date]: 该元素中内容必须为有效日期。
- ❑ custom[integer]: 该元素中内容必须为整数。
- ❑ custom[number]: 该元素中内容必须为有效数值（允许带小数点）。
- ❑ custom[email]: 该元素中内容必须为有效 E-mail 邮件地址。

除了以上本页面中使用到的规则之外，验证插件中还自带如下几个规则。

- ❑ minSize: 该元素中至少必须输入的字符数。
  - ❑ maxSize: 该元素中最多允许输入的字符数。
  - ❑ min: 该元素中输入的数值必须大于等于指定数值。
  - ❑ max: 该元素中输入的数值必须小于等于指定数值。
  - ❑ past: 该元素中输入的日期必须早于或等于某个指定日期。
  - ❑ future: 该元素中输入的日期必须晚于或等于某个指定日期。
  - ❑ maxCheckbox: 最多允许几个复选框处于选取状态。
  - ❑ minCheckbox: 最少需要几个复选框处于选取状态。
  - ❑ equals: 用于确认所验证的元素中的内容必须等于某个指定元素中的内容。可用于确认密码等场合。
  - ❑ telephone: 该元素中内容必须为有效的电话号码。
  - ❑ ipv4: 该元素中内容必须为有效的 IP 地址。
  - ❑ url: 该元素中内容必须为有效的网址。
  - ❑ onlyNumberSp: 该元素中只能输入数字。
  - ❑ onlyLetterSp: 该元素中只能输入字母。
  - ❑ onlyLetterNumber: 提交时该元素中内容只能为字母或数字, 不允许输入其他字符。
  - ❑ ajaxUserCall: 需要开发者修改其中默认设置的 file 属性与 extraData 属性, 用于向某个网址执行 ajax 提交, 验证用户名是否有效。
  - ❑ ajaxUserCallPhp: 需要开发者修改其中默认设置的 file 属性与 extraData 属性, 用于向某个文件执行 ajax 提交, 验证用户名是否有效。可根据文件类型修改这个验证规则的验证名, 例如, 向 asp 文件执行 ajax 提交时, 可改成 ajaxUserCallAsp。
  - ❑ ajaxNameCall: 需要开发者修改其中默认设置的 file 属性, 用于向某个网址执行 ajax 提交, 验证名字是否已被使用。
  - ❑ ajaxNameCallPhp: 需要开发者修改其中默认设置的 file 属性, 用于向某个文件执行 ajax 提交, 验证名字是否已被使用。可根据文件类型修改这个验证规则的验证名, 例如, 向 asp 文件执行 ajax 提交时, 可改成 ajaxNameCallAsp。
- 后四个验证规则用于 ajax 提交, 以下为一个典型的用户登录时所使用的用户名文本框。

```
<input type="text" id="userName" name="userName" maxlength="20"
class="validate[ajax[ajaxUser]]"
```

我们可以使用 ajaxUser 验证规则对用户名进行验证。该验证规则定义如下。

```
"ajaxUser":{
/*file 属性值为后台提交的脚本文件, 插件会提交三个参数:
validateError, validateId, validateValue, 后台脚本直接 request 即可 */
"file": "validateUser.php",
"extraData": "name=eric",
"alertTextOk": "* This username is available",
"alertTextLoad": "* Loading, please wait",
```

```
"alertText": "* This username is already taken"},
```

我们可以将“validateUser.php”修改为任何需要验证的用户后台文件，但是该后台脚本文件必须返回固定格式的 json 数据。官方网站提供的作为示例的 php 文件内容如代码清单 2-4 所示。

代码清单 2-4 官方网站提供的用于验证用户名的 php 文件

---

```
<?php
/* 获取提交值 */
$validateValue=$_GET['fieldValue'];
$validateId=$_GET['fieldId'];
$validateError= "This username is already taken";
$validateSuccess= "This username is available";

/* 返回值 */
$arrayToJs = array();
$arrayToJs[0] = $validateId;

if($validateValue == "duncan"){           // 验证通过
    $arrayToJs[1] = true;                  // 返回 TRUE
    echo json_encode($arrayToJs);        // 返回成功时的 json 数组
}
else// 验证失败
{
    for($x=0;$x<1000000;$x++){
        if($x == 990000){
            $arrayToJs[1] = false;
            echo json_encode($arrayToJs); // 返回失败时的 json 数组
        }
    }
}
}
?>
```

---

开发者可根据需要将该文件名和文件内容修改成其他语言和验证逻辑对应的验证文件。

### 3) 根据需要修改验证方法。

在页面中可以不使用默认的插件，而使用如下代码来修改验证插件对元素内容的验证方法。例如，验证插件的默认验证时机为控件失去焦点时触发验证，用户可以在页面中加入如下代码，在提交控件时进行验证。

```
$(function () {
    $("#form1").validationEngine({
        //inlineValidation: true
        inlineValidation: true, // 修改为 false
        success : false,
        failure:function{callFailFunction{}}
    });
});
```

或者加入如下代码，在其他的时机进行验证。

```
$(function () {
    $("#form1").validationEngine({
        //validationEventTriggers:"focusout",
        validationEventTriggers:"keyup",, // 修改为按键后即执行验证
        success : false,
        failure:function{callFailFunction{}}
    });
});
```

也可以加入如下代码，修改验证错误提示信息与被验证元素的位置关系。

```
$(function () {
    $("#form1").validationEngine({
        //promptPosition: "topRight",// 验证错误提示信息位于元素的右上角

        // 修改为位于元素的右下角，
        // 可设定值为 topLeft, topRight, bottomLeft, centerRight, bottomRight
        validationEventTriggers:"bottomRight",,
        success : false,
        failure:function{callFailFunction{}}
    });
});
```

#### 4) 添加中文版验证错误提示信息文件。

由于从官方网站上下载的压缩包中的验证错误提示信息文件 [在 (下载文件夹) \formValidator.2.1\js\languages] 中，没有中文版的验证错误提示信息文件，因此需要手工添加中文版的验证错误提示信息文件。笔者添加的中文版验证错误提示信息文件如代码清单 2-5 所示。

代码清单 2-5 笔者添加的中文版验证错误提示信息文件

---

```
(function($){
    $.fn.validationEngineLanguage = function(){
    };
    $.validationEngineLanguage = {
        newLang: function(){
            $.validationEngineLanguage.allRules = {
                "required": { // 追加正则表达式作为验证规则
                    "regex": "none",
                    "alertText": "* 必须输入内容 ",
                    "alertTextCheckboxMultiple": "* 必须选择一个选项 ",
                    "alertTextCheckboxes": "* 必须为选取状态 "
                },
                "minSize": {
                    "regex": "none",
                    "alertText": "* 至少必须输入 ",
                    "alertText2": " 个字符 "
```

```

    },
    "maxSize": {
        "regex": "none",
        "alertText": "** 最多允许输入 ",
        "alertText2": " 个字符 "
    },
    "min": {
        "regex": "none",
        "alertText": "** 数字必须大于等于 "
    },
    "max": {
        "regex": "none",
        "alertText": "** 数字必须小于等于 "
    },
    "past": {
        "regex": "none",
        "alertText": "** 日期必须早于 "
    },
    "future": {
        "regex": "none",
        "alertText": "** 日期必须晚于 "
    },
    "maxCheckbox": {
        "regex": "none",
        "alertText": "** 超出允许选取的项目个数 "
    },
    "minCheckbox": {
        "regex": "none",
        "alertText": "** 请至少选择 ",
        "alertText2": " 个选项 "
    },
    "equals": {
        "regex": "none",
        "alertText": "** 输入的内容不一致 "
    },
    "phone": {
        "regex": /^([\+][0-9]{1,3}[\ \-\.])?([()\{1}[0-9]{2,6}[\ \-])?([0-9\-\-\/]{3,20})((x|ext|extension)[ ])?[0-9]{1,4})?$/,
        "alertText": "** 请输入有效的电话号码 "
    },
    "email": {
        "regex": /^([A-Za-z0-9_\-\.\'])+\@([A-Za-z0-9_\-\.\'])+\.[A-Za-z]{2,6})$/,
        "alertText": "** 请输入有效的邮件地址 "
    },
    "integer": {
        "regex": /^[\-\/+]?[0-9]+$/,
        "alertText": "** 请输入一个整数 "
    },
    },

```

```

"number": {
    "regex": /^[\-\.+]?(((0-9)+)([\.,]([0-9]+))
    ?|([\.,]([0-9]+)))?$/ ,
    "alertText": " * 请输入一个数值 "
},
"date": {
    "regex": /^\\d{4}[\\/] (0?[1-9]|1[012])
    [\\/] (0?[1-9]|1[12] [0-9]|3[01])$/ ,
    "alertText": " * 日期格式不正确，必须为 YYYY-MM-DD 格式 "
},
"ipv4": {
    "regex": /^(((01)?[0-9]{1,2})|(2[0-4][0-9])
    |(25[0-5]))[.]{3} (((0-1)?[0-9]{1,2})|(2[0-4][0-9])
    |(25[0-5]))$/ ,
    "alertText": " * 请输入一个有效的 IP 地址 "
},
"url": {
    "regex": /^((https?|ftp):\\/(\\/(((a-z)|\\d|\\.|_|~|
    \\u00A0-\\uD7FF\\uF900-\\uFDCF\\uFDF0-\\uFFEF))|(%[\\da-f]{2})
    |(!\\$&'\\(\\)\\*\\+,;=)|:|@)?(((\\d|[1-9]\\d|1\\d\\d|2[0-4]
    \\d|25[0-5])\\. (\\d|[1-9]\\d|1\\d\\d|2[0-4]\\d|25[0-5])
    \\. (\\d|[1-9]\\d|1\\d\\d|2[0-4]\\d|25[0-5])\\. (\\d|[1-9]\\d|1\\d
    \\d|2[0-4]\\d|25[0-5]))|(((a-z)|\\d|[\\u00A0-\\uD7FF\\uF900-
    \\uFDCF\\uFDF0-\\uFFEF))|(((a-z)|\\d|[\\u00A0-\\uD7FF\\uF900-
    \\uFDCF\\uFDF0-\\uFFEF))([a-z]|\\d|\\.|_|~|[\\u00A0-\\uD7FF
    \\uF900-\\uFDCF\\uFDF0-\\uFFEF))*([a-z]|\\d|[\\u00A0-\\uD7FF
    \\uF900-\\uFDCF\\uFDF0-\\uFFEF)))\\.)+( ([a-z]|[\\u00A0-
    \\uD7FF\\uF900-\\uFDCF\\uFDF0-\\uFFEF))|(((a-z)|[\\u00A0-
    \\uD7FF\\uF900-\\uFDCF\\uFDF0-\\uFFEF))([a-z]|\\d|\\.|_|~|[
    \\u00A0-\\uD7FF\\uF900-\\uFDCF\\uFDF0-\\uFFEF))*([a-z)|[
    \\u00A0-\\uD7FF\\uF900-\\uFDCF\\uFDF0-\\uFFEF)))\\.)?(:
    \\d*)?(\\/(((a-z)|\\d|\\.|_|~|[\\u00A0-\\uD7FF\\uF900-
    \\uFDCF\\uFDF0-\\uFFEF))|(%[\\da-f]{2})|(!\\$&'\\(\\)\\*\\+,;=)
    |:|@)+\\/(((a-z)| \\d|\\.|_|~|[\\u00A0-\\uD7FF\\uF900-
    \\uFDCF\\uFDF0-\\uFFEF))|(%[\\da-f]{2})|(!\\$&'\\(\\)\\*
    \\+,;=)|:|@)*)*)?\\)?(((a-z)|\\d|\\.|_|~|[\\u00A0-
    \\uD7FF\\uF900-\\uFDCF\\uFDF0-\\uFFEF))|(%[\\da-f]{2})|(!\\$&'
    \\(\\)\\*\\+,;=)|:|@)|[\\uE000-\\uF8FF]|\\/|\\/?)*\\)?(\\#(((a-z)|
    \\d|\\.|_|~|[\\u00A0-\\uD7FF\\uF900-\\uFDCF\\uFDF0-\\uFFEF))
    |(%[\\da-f]{2})|(!\\$&'\\(\\)\\*\\+,;=)|:|@)|\\/|\\/?)*)?$/ ,
    "alertText": " * 请输入一个有效的网址 "
},
"onlyNumberSp": {
    "regex": /^[0-9\\ ]+$/ ,
    "alertText": " * 只能输入数字 "
},
"onlyLetterSp": {
    "regex": /^[a-zA-Z\\ ']+$/ ,
    "alertText": " * 只能输入字母 "
},

```

```

        "onlyLetterNumber": {
            "regex": /^[0-9a-zA-Z]+$/,
            "alertText": "** 只能输入数字或字母 "
        },
        "ajaxUserCall": {
            "url": "ajaxValidateFieldUser",
            "extraData": "name=eric",
            "alertText": "** 用户名已被使用 ",
            "alertTextLoad": "** 正在验证, 请稍候 "
        },
        "ajaxUserCallPhp": {
            "url": "phpajax/ajaxValidateFieldUser.php",
            "extraData": "name=eric",
            "alertTextOk": "** 用户名有效 ",
            "alertText": "** * 用户名已被使用 ",
            "alertTextLoad": "** 正在验证, 请稍候 "
        },
        "ajaxNameCall": {
            "url": "ajaxValidateFieldName",
            "alertText": "** 这个名字已被使用 ",
            "alertTextOk": "** 这个名字可以使用 ",
            "alertTextLoad": "** 正在验证, 请稍候 "
        },
        "ajaxNameCallPhp": {
            "url": "phpajax/ajaxValidateFieldName.php",
            "alertText": "** 这个名字已被使用 ",
            "alertTextLoad": "** 正在验证, 请稍候 "
        },
        "validate2fields": {
            "alertText": "** 请输入 HELLO"
        }
    };
}

$.validationEngineLanguage.newLang();
})(jQuery);

```

## 2.2.4 代码剖析

### 1. HTML 5 页面代码

接下来具体看一下该页面的 HTML 5 页面代码部分, 该页面为一个商业订单数据的输入页面, 分为上下两部分, 分别存放在两个 section 元素中, 两个 section 元素的标题分别为“编辑订单信息”与“订单信息一览表”。在第一个 section 元素中, 存放了一个供用户输入订单数据的表单, 在该表单内使用了很多 HTML 5 中新增的表单元素。在第二个 section 元素中, 存放了一张用户输入的订单数据一览表, 页面打开时, 该一览表中显示数据库中保存的所有订单数据, 用户输入一条订单数据并单击保存按钮后, 该表单被提交到服务器端, 用

户输入的这条订单数据被保存到数据库中，然后该页面被刷新，一览表中即时显示用户输入的这条数据。

该页面中所有控件的信息如表 2-1 所示。

表 2-1 页面中所有控件的信息

控件名称	使用元素	显示文字	最大输入位数	说明
“编辑订单信息”标题文字	h5	编辑订单信息	—	
“订单编号”标签	label	订单信息	—	
“订单编号”文本框	input type="text"		8	使用 placeholder 属性显示提示文字“必须输入订单编号”。使用 required 属性在提交时进行必须输入验证。 使用 autofocus 属性在页面打开时自动获取焦点
“订单日期”标签	label	订单日期	—	
“订单日期”文本框	input type="date"		10	使用 placeholder 属性显示提示文字“必须输入一个有效的日期”。 使用 required 属性在提交时进行必须输入验证
“商品编号”标签	label	商品编号	—	
“商品编号”文本框	input type="text"		12	使用 placeholder 属性显示提示文字“必须输入商品编号”。 使用 required 属性在提交时进行必须输入验证
“商标”标签	label	商标		
“商标”文本框	input type="text"		50	
“数量”标签	label	数量		
“数量”文本框	input type="number"	画面打开时内容为“0”	6	使用 placeholder 属性显示提示文字“必须输入一个整数值”
“单价”标签	label	单价		
“单价”文本框	input type="text"	画面打开时内容为“0”	6	使用 placeholder 属性显示提示文字“必须输入一个有效的单价”
“金额”标签	label	金额		
“金额”文本框	input type="text"	画面打开时内容为“0”		只读控件。当数量文本框或单价文本框失去焦点时设置“金额”文本框中内容=数量*单价
“负责人”标签	label	负责人		



(续)

控件名称	使用元素	显示文字	最大输入位数	说明
“负责人”文本框	input type="text"		20	
“负责人 Email”标签	label	负责人 Email		
“负责人 Email”文本框	input type="email"		20	使用 placeholder 属性显示提示文字“请输入一个有效的邮件地址”
新增按钮	asp:Button	新增		
追加按钮	asp:Button	追加		
修改按钮	asp:Button	修改		
删除按钮	asp:Button	删除		
清除按钮	asp:Button	清除		
“订单信息一览表”标题文字	h5	订单信息一览表	—	
订单信息一览表	datagrid			
“订单编号”列	asp:BoundColumn			
“订单日期”列	asp:BoundColumn			
“商品编号”列	asp:BoundColumn			
“商标”列	asp:BoundColumn			
“数量”列	asp:BoundColumn			
“单价”列	asp:BoundColumn			
“金额”列	asp:BoundColumn			
“负责人”列	asp:BoundColumn			
“负责人 Email”列	asp:BoundColumn			
“编辑”列	asp:ButtonColumn			

在查看这段 HTML 5 页面代码之前，要注意的是，本页面中很多文本框使用了一个特殊的 class 属性，在“2.2.3 案例知识点”中已介绍过，这些 class 属性是使用 jQuery 验证器插件对表单中的各元素内容进行验证时所使用的各种验证规则，这里不再赘述。

该页面中的 HTML 5 页面代码如代码清单 2-6 所示。

代码清单 2-6 页面中的 HTML 5 页面代码

```
<%@ Page Title=" 订单信息 " Language="C#" AutoEventWireup="true"
CodeBehind="Default.aspx.cs" Inherits="HTML5TEST.Default" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title> 订单信息 </title>
<body onload="init()">
<form id="form1" method="post" runat="server">
```



[illegible]

```

<section>
<section>
    <h5> 订单信息一览表 </h5>
    <div id="infoTable">
        <asp:datagrid id="datatable"
            runat="server"
            Width="100%"
            BorderWidth="0px"
            AutoGenerateColumns="false"
            CellPadding="1"
            CellSpacing="1"
            AllowPaging="True"
            PageSize="10"
            EnableViewState="true"
            OnPageIndexChanged="datatable_PageIndexChanged"
            OnItemCommand="datatable_ItemCommand">
            <Columns>
                <asp:BoundColumn DataField="code" HeaderText=" 订单编号 "/>
                <asp:BoundColumn DataField="orderDate"
                    HeaderText=" 订单日期 "/>
                <asp:BoundColumn DataField="goodsCode"
                    HeaderText=" 商品编号 "/>
                <asp:BoundColumn DataField="brandName" HeaderText=" 商标 "/>
                <asp:BoundColumn DataField="num" HeaderText=" 数量 "/>
                <asp:BoundColumn DataField="price" HeaderText=" 单价 "/>
                <asp:BoundColumn DataField="money" HeaderText=" 金额 "/>
                <asp:BoundColumn DataField="personName" HeaderText=" 负责人 "/>
                <asp:BoundColumn DataField="email" HeaderText=" 负责人 Email "/>
                <asp:ButtonColumn CommandName="select" HeaderText=" 编辑 "
                    Text=" 编辑 "/>
            </Columns>
            <PagerStyle HorizontalAlign="Right" Width="100%"
                Mode="NumericPages" Position="Bottom" Wrap="false" />
            <SelectedItemStyle BackColor="Blue" ForeColor="White" />
        </asp:datagrid>
    </div>
</section>
</form>
</body>
</html>

```

## 2. 样式代码

在这个页面中，使用了很多 CSS3 样式，下面先对这些 CSS3 样式进行讲解。

在表单部分，使用 ul 列表中嵌套 ul 列表的形式对表单进行布局。其中，内层的 ul 列表用于横向布局，每行显示三个 label 标签与三个 TextBox 文本框，外层的 ul 列表用于竖向布局，将内层的 ul 列表进行竖向排列。这两个 ul 列表的样式代码如下所示。

```
// 外层的 ul 列表用于竖向布局
```

```

ul{
    width:100%;
    display: -moz-box;
    display: -webkit-box;
    -moz-box-orient: vertical;
    -webkit-box-orient:vertical;
    margin:0px;
    padding:0px;
}
// 内层的 ul 列表用于横向布局
ul li ul{
    display: -moz-box;
    display: -webkit-box;
    -moz-box-orient: horizontal;
    -webkit-box-orient:horizontal;
}

```

将每个 li 列表项目元素的 list-style 属性设为 none，使其默认列表项目的项目编号不被显示。代码如下所示。

```

li{
    list-style:none;
}

```

使用 [att^=val] 属性选择器分别控制表单中每个用于放置标签的 li 列表项目元素的样式与每个用于放置文本框的 li 列表项目元素的样式。例如，如下 HTML 代码，表单中所有放置标签的 li 列表项目元素的 id 都是以“title”文字开头的，放置文本框的 li 列表项目元素的 id 都是以“content”文字开头的。

```

<li id="title_1"><label for="tbxCode"> 订单编号 </label></li>
<li id="content_1">
    <input type="text" name="tbxCode" id="tbxCode" maxlength="8"
    class="validate[required]" placeholder="必须输入订单编号" required
    autofocus/>
</li>

```

使用如下样式代码。

```

<li[id^=title]{
    font-size: 12px;
    color: #333333;
    background-color:#E6E6E6;
    text-align:right;
    padding-right:5px;
    width:120px;
}
li[id^=content]{
    height:22;
    background-color:#FAFAFA;
}

```

```

text-align:left;
padding-left:2px;
width:220px;
}

```

使用 E: read-only 伪类选择器控制表单中文本框处于只读状态时的样式，代码如下所示。

```

input:read-only{
    background-color:yellow;
}
// 在Firefox浏览器中需使用-moz-前缀
input:-moz-read-only{
    background-color:yellow;
}

```

使用属性选择器控制所有按钮样式，代码如下所示。

```

input[type="submit"]{
    font-size: 12px;
    width: 68px;
    height: 20px;
    cursor: hand;
    border:none;
    font-family: 宋体;
    background-color:White;
    background-image: url(images/but_bg.gif);
    color: white;
}

```

对于 ASP.NET 的 DataGrid 控件，其样式不在页面上书写，而是在样式代码中使用 CSS3 样式代码统一书写，同时使用 nth-child(even) 选择器与 nth-child(odd) 选择器分别控制一览表中所有偶数行与奇数行的背景色与文字颜色。使用 nth-child(1) 选择器单独控制标题行的背景色与文字颜色。使用 nth-last-child(1) 选择器单独控制一览表中最后一列（编辑按钮列）的宽度。代码如下所示。

```

div#infoTable table tr:nth-child(odd){
    background-color:#E6E6E6;
    color: #333333;
}
div#infoTable table tr:nth-child(even){
    background-color:#fafafa;
    color: black;
}
div#infoTable table tr:nth-child(1){
    background-color:#7088AD;
    color: #FFFFFF;
}
div#infoTable table th:nth-last-child(1){
    width: 15%;
}

```

该页面的完整样式代码如代码清单 2-7 所示。

代码清单 2-7 页面的完整的样式代码

---

```
<style type="text/css">
body {
    margin-left: 0px;
    margin-top: 0px;
}
h5{
    margin:0px;
}
ul{
    width:100%;
    display: -moz-box;
    display: -webkit-box;
    -moz-box-orient: vertical;
    -webkit-box-orient:vertical;
    margin:0px;
    padding:0px;
}
li{
    list-style:none;
}
ul li ul{
    display: -moz-box;
    display: -webkit-box;
    -moz-box-orient: horizontal;
    -webkit-box-orient:horizontal;
}
header{
    font-size: 14px;
    font-weight: bold;
    color:white;
    background-color:#7088AD;
    text-align:left;
    padding-left:10px;
    display:block;
    width:100%;
}
li[id^=title]{
    font-size: 12px;
    color: #333333;
    background-color:#E6E6E6;
    text-align:right;
    padding-right:5px;
    width:120px;
}
li[id^=content]{
    height:22;
```

```

        background-color:#FAFAFA;
        text-align:left;
        padding-left:2px;
        width:220px;
    }
    span{
        color: #ff0000;
    }
    input{
        width: 95%;
        border-style: solid;
        border-top-color: #426C7C;
        border-right-color: #CCCCCC;
        border-bottom-color: #CCCCCC;
        border-left-color: #426C7C;
        border-width: 1px;
        border:1px solid #0066cc;
        height: 18px;
    }
    input:read-only{
        background-color:yellow;
    }
    input:-moz-read-only{
        background-color:yellow;
    }
    input#tbxNum{
        text-align:right;
    }
    input#tbxPrice{
        text-align:right;
    }
    input#tbxMoney{
        text-align:right;
    }
    #tbxBrand,#tbxPerson,#tbxProduct{
        width:20px;
    }
    div{
        text-align:right;
    }
    div#buttonDiv{
        width:100%;
    }
    input[type="submit"]{
        font-size: 12px;
        width: 68px;
        height: 20px;
        cursor: hand;
        border:none;
        font-family: 宋体;
    }

```



```

        background-color:White;
        background-image: url(images/but_bg.gif);
        color: white;
    }
    div#infoTable{
        overflow:auto;
        width:100%;
        height:100%;
    }
    div table{
        width:100%;
        background-color:white;
        cellpadding:1;
        cellspacing:1;
        font-size: 12px;
        text-align: center;
    }
    div table th{
        height:22;
        background-color:#7088AD;
        color: #FFFFFF;
        width:8%;
    }
    div table tr{
        height:30;
    }
    div#infoTable table tr:nth-child(odd){
        background-color:#E6E6E6;
        color: #333333;
    }
    div#infoTable table tr:nth-child(even){
        background-color:#fafafa;
        color: black;
    }
    div#infoTable table tr:nth-child(1){
        background-color:#7088AD;
        color: #FFFFFF;
    }
    div#infoTable table th:nth-last-child(1){
        width: 15%;
    }
</style>

```

---

### 3. jQuery 脚本代码

下面来集中看一下本页面中使用了jQuery验证器的几个HTML元素的页面代码，如下所示。

```

<input type="text" id="tbxCode" name="tbxCode" maxlength="8"
class="validate[required]" placeholder="必须输入一个不存在的订单编号"

```

```
autofocus required/>
<input type="date" id="tbxDate" name="tbxDate" maxlength="10"
class="validate[required,custom[date]]"
placeholder="必须输入一个有效的日期" required/>
<input type="text" id="tbxGoodsCode" name="tbxGoodsCode" maxlength="12"
class="validate[required]" placeholder="必须输入商品编号" required/>
  <input type="number" id="tbxNum" name="tbxNum" maxlength="6" value="0"
class="validate[required,custom[integer]]"
placeholder="必须输入一个整数值" />
<input type="text" id="tbxPrice" name="tbxPrice" maxlength="6" value="0"
class="validate[required,custom[number]]"
placeholder="必须输入一个有效的单价" />
<input type="email" id="tbxEmail" name="tbxEmail" maxlength="20"
class="validate[optional,custom[email]]"
placeholder="请输入一个有效的邮件地址" />
```

最后，除了根据后台服务器端返回的变量进行页面设置的脚本代码之外，本页面中还有两个小的 JavaScript 函数，它们的功能是当数量文本框或单价文本框失去焦点时将金额文本框中的内容设为数量文本框中的数量值 \* 单价文本框中的单价值。

目前该页面中的 JavaScript 脚本代码如代码清单 2-8 所示。

代码清单 2-8 页面中的 JavaScript 脚本代码

---

```
<link rel="stylesheet" href="Styles/validationEngine.jquery.css"
type="text/css" media="screen"/>
<script src="Scripts/jquery-1.5.1.min.js" type="text/javascript">
</script>
<script src="Scripts/jquery.validationEngine-cn.js"
type="text/javascript"></script>
<script src="Scripts/jquery.validationEngine.js" type="text/javascript">
</script>
<script>
  $(function () {
    $("#form1").validationEngine();
    $("#tbxNum").blur(function () {
      var num=parseInt($("#tbxNum").val());
      var price=parseFloat($("#tbxPrice").val());
      if (isNaN(num*price))
        $("#tbxMoney").val("0");
      else
        $("#tbxMoney").val(Math.round(num*price * 100, 0) / 100);
    });
    $("#tbxPrice").blur(function () {
      var num=parseInt($("#tbxNum").val());
      var price=parseFloat($("#tbxPrice").val());
      if (isNaN(num*price))
        $("#tbxMoney").val("0");
      else
        $("#tbxMoney").val(Math.round(num*price * 100, 0) / 100);
    });
  });
</script>
```

```
});
});
```

#### 4. ASP.NET 服务器端代码

最后来看一下这个页面的服务器端的 ASP.NET 代码部分。

在该页面中，使用了 HTML 5 中新增的 date 元素、number 元素和 email 元素，代码如下所示。

```
<input type="date" id="tbxDate" name="tbxDate" maxlength="10"
class="validate[required,custom[date]]"
placeholder="必须输入一个有效的日期" required/>
<input type="number" id="tbxNum" name="tbxNum" maxlength="6" value="0"
class="validate[required,custom[integer]]"
placeholder="必须输入一个整数值" />
<input type="email" id="tbxEmail" name="tbxEmail" maxlength="20"
class="validate[optional,custom[email]]"
placeholder="请输入一个有效的邮件地址" />
```

对于这些元素，在目前最新版本的 ASP.NET 服务器端控件中，没有与之相对应的服务器端控件，所以在 HTML 页面代码中，不能书写成“<asp:TextBox>”这种采用服务器端控件的形式。那么，为了在服务器端获取这几个元素的输入值，能不能使用如下所示的将 runat 的属性值设为 server 的方法呢？

```
<input type="date" id="tbxDate" name="tbxDate" maxlength="10"
class="validate[required,custom[date]]"
placeholder="必须输入一个有效的日期" required runat="server"/>
<input type="number" id="tbxNum" name="tbxNum" maxlength="6" value="0"
class="validate[required,custom[integer]]" placeholder="必须输入一个整数值"
runat="server"/>
<input type="email" id="tbxEmail" name="tbxEmail" maxlength="20"
class="validate[optional,custom[email]]"
placeholder="请输入一个有效的邮件地址" runat="server"/>
```

在页面中将这几个元素的 runat 属性值设为 server，然后在浏览器中运行该页面，浏览器显示一个报错页面，如图 2-10 所示。

出现这个服务器端返回的分析器错误的原因是，将这几个 HTML 5 中新增元素的 runat 属性值设为 server 后，ASP.NET 服务器会将这几个元素视为服务器端控件，而在目前最新版本的 ASP.NET 中，是没有这几个服务器端控件的。

那么，应该怎样在服务器端获取用户通过这几个元素输入的值呢？

首先，将这几个新增元素的 runat 属性值取消，即将这几个元素作为客户端元素来处理，代码如下所示。

```
<input type="date" id="tbxDate" name="tbxDate" maxlength="10"
class="validate[required,custom[date]]"
placeholder="必须输入一个有效的日期" required />
```

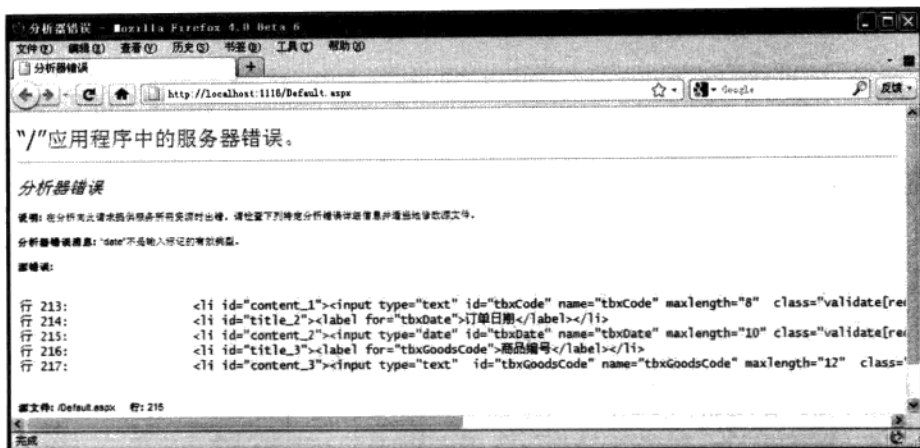


图 2-10 将 HTML 5 中新增元素的 runat 属性值设为 server 后浏览器报错

然后, 在服务器端代码中 (本例中使用 C# 语言) 加入 `Request.Form["tbxDate"].ToString()` 方法。例如, 在服务器端的代码中, 加上如下这段代码。

```
protected void Page_Load(object sender, EventArgs e)
{
    If (Page.IsPostBack)
    {
        Response.Write(Request.Form["tbxDate"].ToString());
    }
}
```

在页面上表单的各元素中输入有效数据, 在“订单日期”文本框中输入“2009-06-01”, 如图 2-11 所示。

在“订单日期”文本框  
中输入“2009-06-01”

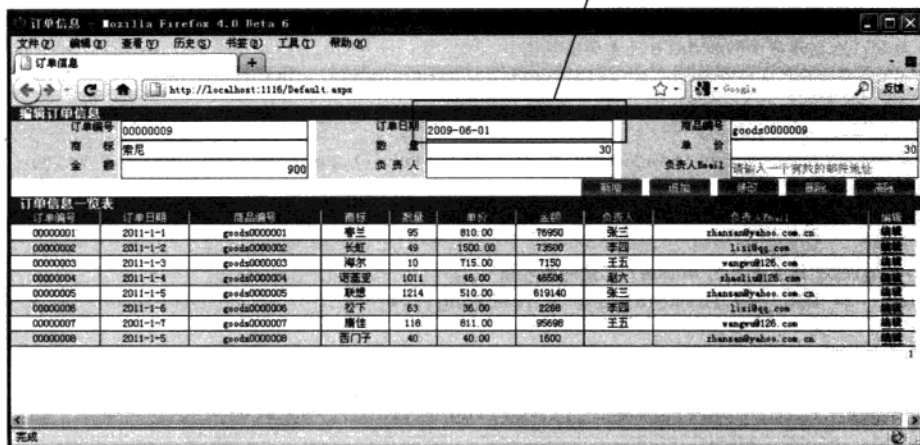


图 2-11 在“订单日期”文本框中输入“2009-06-01”

然后单击新增按钮，提交表单后浏览器的显示效果如图 2-12 所示。

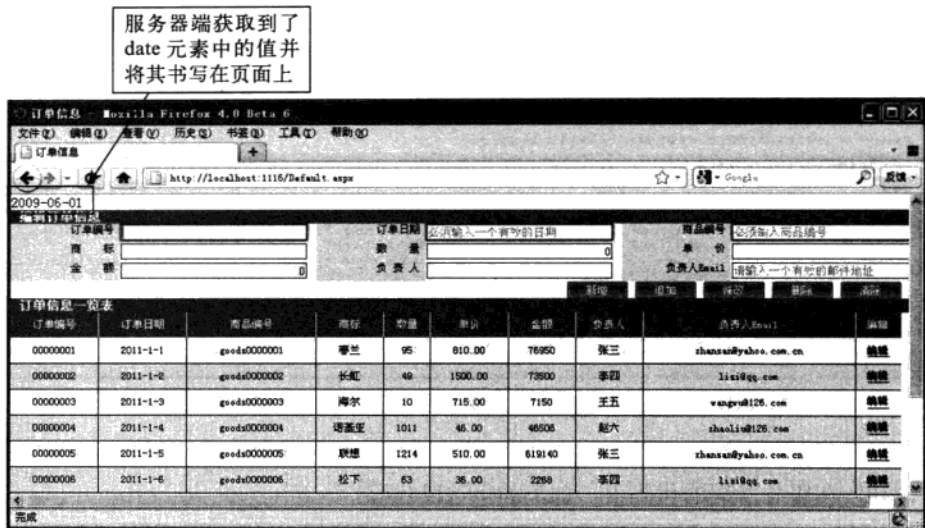


图 2-12 提交表单后的页面显示效果

在这个服务器端返回的结果页面上可以看到，ASP.NET 服务器端获取到了用户在 date 元素中输入的“2009-06-01”，并将其显示在返回的页面上。

在这个基础上，我们可以采用如下所示的代码，将获取到的用户通过 HTML 5 新增元素输入的内容保存到数据库中。

```
string SqlStr;
SqlStr = "insert into orders ";
SqlStr += "values('";
SqlStr += Request.Form["tbxCODE"].ToString().Trim().Replace("'", "'")
+ "','";

// 向 SqlStr 语句中添加 date 元素的输入值
SqlStr += "'" + Request.Form["tbxDate"].ToString() + "','";

SqlStr += "'"
+ Request.Form["tbxGoodsCode"].ToString().Trim().Replace("'", "'") + "','";
SqlStr += "'"
+ Request.Form["tbxBrandName"].ToString().Trim().Replace("'", "'") + "','";

// 向 SqlStr 语句中添加 number 元素的输入值
SqlStr += Request.Form["tbxNum"].ToString() + "','";

SqlStr += Request.Form["tbxPrice"].ToString() + "','";
SqlStr += "'"
+ Request.Form["tbxPersonName"].ToString().Trim().Replace("'", "'")
+ "','";
```

```
// 向 SqlStr 语句中添加 email 元素的输入值
SqlStr += " "
+ Request.Form["tbxEmail"].ToString().Trim().Replace("'", "'") + " ";

// 执行 SQL 语句, 向数据库中追加数据
SuccessFlag = this.ExecSingleSql(SqlStr);
```

此外还有一点需要补充, 因为页面中各文本框控件均直接使用了客户端元素 (input 元素), 所以不能使用 ASP.NET 中的 ViewState 机制来保存提交前各元素中的输入内容, 需要在服务器端使用如下所示的代码, 利用服务器端变量来统一获取各元素中的输入内容。

```
public partial class Default : System.Web.UI.Page
{
    public string Code;
    public string date;
    public string GoodsCode;
    public string brandName;
    public string num="0";
    public string price="0";
    public string money="0";
    public string PersonName;
    public string Email;
    protected void Page_Load(object sender, EventArgs e)
    {
        if (Page.IsPostBack)
        {
            this.Code = Request.Form["tbxCode"].ToString();
            this.date = Request.Form["tbxDate"].ToString();
            this.GoodsCode = Request.Form["tbxGoodsCode"].ToString();
            this.brandName = Request.Form["tbxBrandName"].ToString();
            this.num = Request.Form["tbxNum"].ToString();
            this.price = Request.Form["tbxPrice"].ToString();
            double temp = double.Parse(this.num) * double.Parse(this.price);
            this.money = Math.Round(temp, 2).ToString();
            this.PersonName = Request.Form["tbxPersonName"].ToString();
            this.Email = Request.Form["tbxEmail"].ToString();
        }
    }
}
```

然后在 JavaScript 脚本代码中, 在装载页面时将表单中的各元素内容设为提交前保存在服务器端变量中的各项内容, 代码如下所示。

```
$(function () {
    $("#tbxCode").val("<%=Code%>");
    $("#tbxDate").val("<%=date%>");
    $("#tbxGoodsCode").val("<%=GoodsCode%>");
    $("#tbxBrandName").val("<%=brandName%>");
    $("#tbxNum").val("<%=num%>");
```

```

$("#tbxPrice").val("<%=price%>");
$("#tbxMoney").val("<%=money%>");
$("#tbxPersonName").val("<%=PersonName%>");
$("#tbxEmail").val("<%=Email%>");
});

```

接下来, 在查看完整的 ASP.NET 服务器端脚本代码之前, 先查看一下在该页面中单击各个按钮以及对一览表执行某个动作后服务器端所应执行的处理, 如表 2-2 所示。

表 2-2 该页面中各个按钮及其功能

执行动作	功能
单击新增按钮	将表中各输入控件的内容清空, 用于将表单从修改数据状态改变为添加数据状态
单击追加按钮	将用户在表单中输入的数据追加到数据库中
单击修改按钮	将用户对某条数据所做的修改保存到数据库中
单击删除按钮	将用户选择的某条数据从数据库中删除
单击清除按钮	将表单中除了“订单编号”之外的其他各控件内容清空, 使用户在修改数据时重新输入各控件内的数据
单击一览表中每条数据的编辑按钮	在表的各文本框中显示用户所选择数据的对应字段的内容
单击一览表右下部的页号链接	在一览表中显示对应页中的内容

再来看一下为了在这个应用程序中使用 SQLServer Express 数据库, 而在 web.config 配置文件中添加的连接字符串。读者需要根据自己的数据库类型 (可以使用 MySQL 和 Oracle 等其他数据库) 及其存放位置做相应的修改。

```

<configuration>
  <connectionStrings>
    <add name="dbConnection"
      connectionString="Data Source=lln\sqlexpress;
      Initial Catalog=HTML5TEST;Integrated Security=True"
      providerName="System.Data.SqlClient" />
  </connectionStrings>

```

下面来看一下该页面完整的 ASP.NET 服务器端脚本代码, 如代码清单 2-9 所示。

代码清单 2-9 页面中完整的 ASP.NET 服务器端脚本代码

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Configuration;
using System.Data;
using System.Data.SqlClient;
namespace HTML5TEST

```

```

{
public partial class Default : System.Web.UI.Page
{
    private System.Data.SqlClient.SqlConnection SqlCon;
    private System.Data.SqlClient.SqlDataAdapter DataAdapter;
    private System.Data.SqlClient.SqlCommand Command;
    private int myErrorNumber = 0; // 数据库返回的错误号
    private string myErrorMessage = ""; // 数据库返回的错误信息
    string Constr; // 连接字符串

    // 用于保存表单中各输入控件内容的变量
    public string Code;
    public string date;
    public string GoodsCode;
    public string brandName;
    public string num="0";
    public string price="0";
    public string money="0";
    public string PersonName;
    public string Email;

    // 设置订单编号文本框是否为只读状态
    // 修改与删除状态时为只读状态, 否则为可读写状态
    public bool codeRead=false;

    private bool SuccessFlag; // 数据库操作是否成功

    // 装载页面
    protected void Page_Load(object sender, EventArgs e)
    {
        // 从 web.config 文件中读取连接字符串
        Constr = System.Configuration.ConfigurationManager.
            ConnectionStrings["dbConnection"].ToString();

        // 设置 SQL Server 数据库连接对象
        this.SqlCon = new System.Data.SqlClient.SqlConnection();
        this.SqlCon.ConnectionString = Constr;

        // 连接数据库
        SuccessFlag = OpenDBConnection();
        if (SuccessFlag == false)
        {
            this.ShowError("数据库连接失败。");
        }
        else
        {
            // 从数据库中读取所有订单数据, 装入一览表
            this.datatable.DataSource = CreateDataSource();
            this.datatable.DataBind();
        }
    }
}

```



```

if (Page.IsPostBack)
{
    // 用变量来保存用户在各表单元素中的输入内容
    this.Code = Request.Form["tbxCCode"].ToString();
    this.date = Request.Form["tbxCDate"].ToString();
    this.GoodsCode = Request.Form["tbxGoodsCode"].ToString();
    this.brandName = Request.Form["tbxBrandName"].ToString();
    this.num = Request.Form["tbxNum"].ToString();
    this.price = Request.Form["tbxPrice"].ToString();

    // 设置金额文本框中内容 = 数量 * 单价
    double temp;
    temp = double.Parse(this.num) * double.Parse(this.price);
    this.money = Math.Round(temp, 2).ToString();

    this.PersonName = Request.Form["tbxPersonName"].ToString();
    this.Email = Request.Form["tbxEmail"].ToString();
}
}

// 单击新增按钮
protected void btnNew_Click(object sender, EventArgs e)
{
    this.Code = string.Empty;
    this.dataInit(); // 清空表单元素中的内容

    // 将一览表中当前选取数据变为非选取状态
    this.datatable.SelectedIndex = -1;

    this.btnUpdate.Enabled = false; // 修改按钮变为禁止单击状态
    this.btnDelete.Enabled = false; // 删除按钮变为禁止单击状态
    this.btnClear.Enabled = false; // 清除按钮变为禁止单击状态
    this.btnAdd.Enabled = true; // 追加按钮变为允许单击状态
}

// 单击追加按钮
protected void btnAdd_Click(object sender, EventArgs e)
{
    DataSet ds;
    string SqlStr;

    // 查看输入的订单编号是否已存在
    SqlStr = "select count(*) count from orders where code=" +
        strFormat(Request.Form["tbxCCode"].ToString()) + "'";
    ds = this.GetDataFromDB(SqlStr);
    if (ds == null)
    {
        this.ShowError("数据添加失败。");
        return;
    }

    // 订单编号已存在
    if (Int32.Parse(ds.Tables[0].Rows[0]["count"].ToString()) > 0)

```

```

{
    this.ShowError("输入的订单编号在数据库中已存在。");
    return;
}

// 追加数据
SqlStr = "insert into orders ";
SqlStr += "values('";
+ strFormat(Request.Form["tbxCode"].ToString()) + "','";
SqlStr += "'"+Request.Form["tbxDate"].ToString() + "','";
SqlStr += "'";
+ strFormat(Request.Form["tbxGoodsCode"].ToString()) + "','";
SqlStr += "'";
+ strFormat(Request.Form["tbxBrandName"].ToString()) + "','";
SqlStr += Request.Form["tbxNum"].ToString() + "','";
SqlStr += Request.Form["tbxPrice"].ToString() + "','";
SqlStr += "'";
+ strFormat(Request.Form["tbxPersonName"].ToString()) + "','";
SqlStr += "'";
+ strFormat(Request.Form["tbxEmail"].ToString()) + "'";
SuccessFlag = this.ExecSingleSql(SqlStr);
if (SuccessFlag == false) // 添加数据失败
{
    this.ShowError("数据添加失败。");
}
else // 添加数据成功
{
    this.codeRead = true; // 订单编号文本框变为只读状态
    this.btnUpdate.Enabled = true; // 修改按钮变为允许单击状态
    this.btnDelete.Enabled = true; // 删除按钮变为允许单击状态
    this.btnClear.Enabled = true; // 清除按钮变为允许单击状态
    this.btnAdd.Enabled = false; // 追加按钮变为禁止单击状态

    // 刷新一览表, 显示被追加的数据
    this.datatable.DataSource = CreateDataSource();
    this.datatable.DataBind();
}
}

// 单击修改按钮
protected void btnUpdate_Click(object sender, EventArgs e)
{
    string SqlStr;
    // 在数据库中修改数据
    SqlStr = "update orders ";
    SqlStr += "set orderDate='";
    + Request.Form["tbxDate"].ToString() + "','";
    SqlStr += "goodsCode='";
    + strFormat(Request.Form["tbxGoodsCode"].ToString()) + "','";
    SqlStr += "brandName='";
    + strFormat(Request.Form["tbxBrandName"].ToString()) + "','";

```

```

SqlStr += "num=" + Request.Form["tbxNum"].ToString() + ",";
SqlStr += "price=" + Request.Form["tbxPrice"].ToString() + ",";
SqlStr += "personName='"
+ strFormat(Request.Form["tbxPersonName"].ToString()) + "',";
SqlStr += "email='"
+ strFormat(Request.Form["tbxEmail"].ToString()) + "' ";
SqlStr += "where code='"
+ strFormat(Request.Form["tbxCode"].ToString()) + "'";
SuccessFlag = this.ExecSingleSql(SqlStr);
if (SuccessFlag == false)// 数据修改失败
{
    this.ShowError(" 数据修改失败。");
}
else// 数据修改成功
{
    this.codeRead = true;// 订单编号文本框保持只读状态
    // 刷新一览表, 显示修改后数据
    this.datatable.DataSource = CreateDataSource();
    this.datatable.DataBind();
}
}

// 单击删除按钮
protected void btnDelete_Click(object sender, EventArgs e)
{
    // 从数据库中删除数据
    string SqlStr;
    SqlStr = "delete from orders ";
    SqlStr += "where code='"
+ strFormat(Request.Form["tbxCode"].ToString()) + "'";
    SuccessFlag = this.ExecSingleSql(SqlStr);
    if (SuccessFlag == false)// 删除数据失败
    {
        this.ShowError(" 数据删除失败。");
    }
    else// 删除数据成功
    {
        // 刷新一览表
        this.datatable.DataSource = CreateDataSource();
        this.datatable.DataBind();
        this.codeRead = false;// 订单编号文本框变为可读写状态

        // 将一览表中当前选取数据变为非选取状态
        this.datatable.SelectedIndex = -1;
        this.Code = string.Empty;// 清除订单编号文本框中的内容
        dataInit();// 清除表单中其他各输入元素中的内容
        this.btnUpdate.Enabled = false;// 修改按钮变为禁止单击状态
        this.btnDelete.Enabled = false;// 删除按钮变为禁止单击状态
        this.btnClear.Enabled = false;// 清除按钮变为禁止单击状态
        this.btnAdd.Enabled = true;// 追加按钮变为允许单击状态
    }
}

```

```

    }
}
// 单击清除按钮
protected void btnClear_Click(object sender, EventArgs e)
{
    // 订单编号文本框保持只读状态
    this.codeRead = true;
    // 清除页面中除了订单编号文本框之外的其他所有文本框中的内容
    this.dataInit();
}
// 单击一览表某条数据的编辑按钮
protected void datatable_ItemCommand(object source,
DataGridCommandEventArgs e)
{
    if (e.CommandName == "select")
    {
        // 表单中各文本框中的内容被设定为选取数据的各项对应内容
        this.Code = e.Item.Cells[0].Text;
        this.date = e.Item.Cells[1].Text;
        this.GoodsCode = e.Item.Cells[2].Text;
        if (e.Item.Cells[3].Text == "&nbsp;")
            this.brandName = "";
        else
            this.brandName = e.Item.Cells[3].Text;
        this.num = e.Item.Cells[4].Text;
        this.price = e.Item.Cells[5].Text;
        // 设置金额文本框中内容=数量*单价
        double num= double.Parse(this.num);
        double price= double.Parse(this.price);
        double temp= num * price;
        this.money = Math.Round(temp, 2).ToString();
        if (e.Item.Cells[7].Text == "&nbsp;")
            this.PersonName = "";
        else
            this.PersonName = e.Item.Cells[7].Text;
        if (e.Item.Cells[8].Text == "&nbsp;")
            this.Email = "";
        else
            this.Email = e.Item.Cells[8].Text;
        this.codeRead = true; // 订单编号文本框变为只读状态
        this.btnUpdate.Enabled = true; // 修改按钮变为允许单击状态
        this.btnDelete.Enabled = true; // 删除按钮变为允许单击状态
        this.btnClear.Enabled = true; // 清除按钮变为允许单击状态
        this.btnAdd.Enabled = false; // 追加按钮变为允许单击状态
    }
}
// 单击一览表右下角的页号进行翻页
protected void datatable_PageIndexChanged(object source,
DataGridPageChangedEventArgs e)
{

```

```

        this.datatable.CurrentPageIndex = e.NewPageIndex;
        this.datatable.DataSource = CreateDataSource();
        this.datatable.DataBind();
    }
    // 清空一览表中除订单编号文本框之外其他所有文本框中的内容
    private void dataInit()
    {
        this.date = string.Empty;
        this.GoodsCode = string.Empty;
        this.brandName = string.Empty;
        this.num = "0";
        this.price = "0";
        this.money = "0";
        this.PersonName = string.Empty;
        this.Email = string.Empty;
    }
    // 连接数据库
    private bool OpenDBConnection()
    {
        try
        {
            this.SqlCon.Open();
            return true;
        }
        catch (SqlException e)
        {
            this.myErrorNumber = e.Number;
            this.myErrorMessage = e.Message;
            return false;
        }
        catch (Exception e)
        {
            this.myErrorNumber = e.GetHashCode();
            this.myErrorMessage = e.Message;
            return false;
        }
    }
    // 在页面中显示错误信息
    private void ShowError(string myErrorMessage)
    {
        string strErrMsg;
        if (!this.myErrorMessage.Equals(""))
            strErrMsg = this.myErrorNumber + ":" + this.myErrorMessage;
        else
            strErrMsg = myErrorMessage;
        Page.ClientScript.RegisterStartupScript(Page.GetType(), "",
            "<script language='JavaScript'>alert('" + strErrMsg
            + "');</script>");
    }
    // 获取所有订单数据

```

```

private DataView CreateDataSource()
{
    DataSet ds;
    DataView dv;
    DataTable dt = new DataTable();
    DataRow dr;
    string SqlStr;
    double temp;
    dt.Columns.Add(new DataColumn("code"));
    dt.Columns.Add(new DataColumn("orderDate"));
    dt.Columns.Add(new DataColumn("goodsCode"));
    dt.Columns.Add(new DataColumn("brandName"));
    dt.Columns.Add(new DataColumn("num"));
    dt.Columns.Add(new DataColumn("price"));
    dt.Columns.Add(new DataColumn("money"));
    dt.Columns.Add(new DataColumn("personName"));
    dt.Columns.Add(new DataColumn("email"));
    SqlStr = "select * from orders order by code";
    ds = this.GetDataFromDB(SqlStr);
    if (ds == null)
    {
        dv = null;
        dt = null;
        return dv;
    }
    String orderDate;
    for (int i = 0; i < ds.Tables[0].Rows.Count; i++)
    {
        dr = dt.NewRow();
        dr[0] = ds.Tables[0].Rows[i]["code"].ToString();
        orderDate = ds.Tables[0].Rows[i]["orderDate"].ToString();
        dr[1] = DateTime.Parse(orderDate).ToString("yyyy-MM-dd");
        dr[2] = ds.Tables[0].Rows[i]["goodsCode"].ToString();
        dr[3] = ds.Tables[0].Rows[i]["brandName"].ToString();
        dr[4] = ds.Tables[0].Rows[i]["num"].ToString();
        dr[5] = ds.Tables[0].Rows[i]["price"].ToString();
        temp = double.Parse(ds.Tables[0].Rows[i]["num"].ToString())
            * double.Parse(ds.Tables[0].Rows[i]["price"].ToString());
        dr[6] = Math.Round(temp, 2).ToString();
        dr[7] = ds.Tables[0].Rows[i]["personName"].ToString();
        dr[8] = ds.Tables[0].Rows[i]["email"].ToString();
        dt.Rows.Add(dr);
    }
    dv = new DataView(dt);
    return dv;
}
// 根据 SQL 语句从数据库中获取数据
private System.Data.DataSet GetDataFromDB(string SqlStr)
{
    DataSet ds;

```

```

        try
        {
            ds = new DataSet();
            this.DataAdapter = new SqlDataAdapter(SqlStr,
            this.SqlCon);
            this.DataAdapter.Fill(ds);
            return ds;
        }
        catch (SqlException e)
        {
            this.myErrorNumber = e.Number;
            this.myErrorMessage = e.Message;
            return null;
        }
        catch (Exception e)
        {
            this.myErrorNumber = e.GetHashCode();
            this.myErrorMessage = e.Message;
            return null;
        }
    }
}
// 根据 SQL 语句执行增删改操作
private bool ExecSingleSql(string SqlStr)
{
    try
    {
        this.Command = new SqlCommand(SqlStr);
        this.Command.Connection = this.SqlCon;
        this.Command.ExecuteNonQuery();
        return true;
    }
    catch (SqlException e)
    {
        this.myErrorNumber = e.Number;
        this.myErrorMessage = e.Message;
        return false;
    }
    catch (Exception e)
    {
        this.myErrorNumber = e.GetHashCode();
        this.myErrorMessage = e.Message;
        return false;
    }
}
}
Private String strFormat(String str)
{
    return str.Trim().Replace("'", "");
}
}
}

```

最后, 由于在服务器端使用了一个 codeRead 变量来控制订单编号文本框的只读状态, 当表单处于修改或删除数据状态时该变量为 True, 订单编号文本框为只读状态, 当表单处于新增数据状态时该变量为 False, 该文本框为可读写状态, 所以在 JavaScript 脚本中, 添加装载页面时的代码, 根据该变量来设置订单编号文本框的只读状态, 代码如下所示。

```
<%if (codeRead)
{
    $( "#tbxCode" ).attr( "ReadOnly", "ReadOnly" );
}
else
{
    $( "#tbxCode" ).removeAttr( 'ReadOnly' );
}
%>
```

现在该页面的完整 JavaScript 脚本代码如代码清单 2-10 所示。

代码清单 2-10 页面的完整 JavaScript 脚本代码

---

```
<link rel="stylesheet" href="Styles/validationEngine.jquery.css"
type="text/css" media="screen"/>
<script src="Scripts/jquery-1.5.1.min.js" type="text/javascript"></script>
<script src="Scripts/jquery.validationEngine-cn.js"
type="text/javascript"></script>
<script src="Scripts/jquery.validationEngine.js"
type="text/javascript"></script>
<script>
    $(function () {
        $( "#form1" ).validationEngine();
        $( "#tbxCode" ).val( "<%=Code%>" );
        $( "#tbxDate" ).val( "<%=date%>" );
        $( "#tbxGoodsCode" ).val( "<%=GoodsCode%>" );
        $( "#tbxBrandName" ).val( "<%=brandName%>" );
        $( "#tbxNum" ).val( "<%=num%>" );
        $( "#tbxPrice" ).val( "<%=price%>" );
        $( "#tbxMoney" ).val( "<%=money%>" );
        $( "#tbxPersonName" ).val( "<%=PersonName%>" );
        $( "#tbxEmail" ).val( "<%=Email%>" );
        <%if (codeRead)
        {
            $( "#tbxCode" ).attr( "ReadOnly", "ReadOnly" );
        }
        else
        {
            $( "#tbxCode" ).removeAttr( 'ReadOnly' );
        }
        %>
        $( "#tbxNum" ).blur(function () {
            var num= parseInt( $( "#tbxNum" ).val() );
```



```

        var price= parseFloat($("#tbxPrice").val());
        if (isNaN(num*price))
            $("#tbxMoney").val("0");
        else
            $("#tbxMoney").val(Math.round(num * price * 100, 0) / 100);
    });
    $("#tbxPrice").blur(function () {
        var num= parseInt($("#tbxNum").val());
        var price= parseFloat($("#tbxPrice").val());
        if (isNaN(num*price))
            $("#tbxMoney").val("0");
        else
            $("#tbxMoney").val(Math.round(num * price * 100, 0) / 100);
    });
    $("#btnNew").click(function () {
        $('input[id^="tbx"]').removeClass();
    })
    $("#btnDelete").click(function () {
        $('input[id^="tbx"]').removeClass();
    });
});
</script>

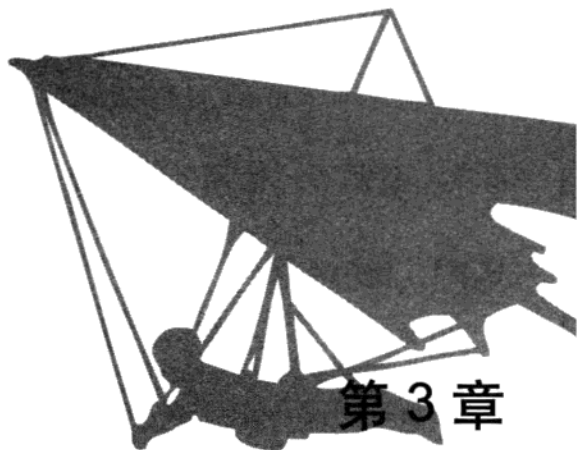
```

---

## 2.3 本章小结

本章首先通过一个利用 HTML 5 中的结构元素制作 Web 应用程序中的菜单的案例来阐述如何使用这些结构元素制作一个语义清晰、结构分明的菜单页面。接下来，通过一个综合使用 HTML 5 中的结构元素和表单元素、jQuery 验证器与 ASP.NET 制作 HTML 5 版本的 Web 应用程序的案例来阐述如何使用 HTML 5 中的结构元素搭建 Web 应用程序的框架结构，如何对用户 HTML 5 新增的表单元素中输入的内容进行验证，以及如何让服务器端能够正确获取到这些用户在新增表单元素中输入的内容并将其保存在数据库中。

下一章将通过一些案例来阐述如何使用 HTML 5 新增的 canvas 元素与 Canvas API 在页面上绘制图形、图像及动画，以及如何利用 Canvas API 制作 Web 页面上的小游戏。



## 第3章

# 使用 canvas 元素绘制图形、 图像与动画

### 本章内容

- 案例 5：使用 canvas 元素绘制美丽的花朵
- 案例 6：使用 canvas 元素绘制指针式动画时钟
- 案例 7：使用 canvas 元素制作简单小游戏
- 案例 8：使用 canvas 元素绘制图像放大镜
- 案例 9：用动画的形式装载图像
- 案例 10：将彩色照片转换成黑白照片
- 本章小结

在 HTML 5 中, canvas 元素是一个非常重要的元素, 用于完成网页中各种图形、图像与动画的绘制工作。本章通过几个利用 canvas 元素在网页上进行图形、图像与动画绘制的案例, 来阐述如何利用该元素来完成网页上各种图形、图像与动画的绘制工作。

## 3.1 案例 5: 使用 canvas 元素绘制美丽的花朵

### 3.1.1 案例概述

在数学中, 有三叶玫瑰线(方程为  $\rho = a \sin(3\theta)$ ) 和四叶玫瑰线(方程为  $\rho = a \sin(2\theta)$ ) 等曲线, 这些曲线的极坐标方程很简单, 基本形式均为  $\rho = a \sin(n\theta)$ , 即任意一点的极半径  $\rho$  是角度  $\theta$  的函数, 其直角坐标方程为:  $x = a \sin(n\theta) \cos(\theta)$ ,  $y = a \sin(n\theta) \sin(\theta)$ 。

在程序中控制角度  $\theta$ , 使其从 0 变化到  $2\pi$ , 描出极半径  $\rho$  所对应的点, 就可以绘制出非常漂亮的玫瑰线。当然,  $n$  值不同, 绘制出的玫瑰线的形状也不相同。

另外, 将方程稍作变化, 还可以绘制出许多其他有花朵效果的漂亮曲线。本节将向读者介绍如何使用 HTML 5 中的 canvas 元素的图形绘制功能, 在网页中绘制出漂亮的曲线, 根据这些曲线的视觉效果, 分别将其称为蓬莱菊、令箭荷花与大丽花。

### 3.1.2 页面显示效果

首先来看一下案例页面在浏览器中的显示效果, 该页面在打开时的显示效果如图 3-1 所示。

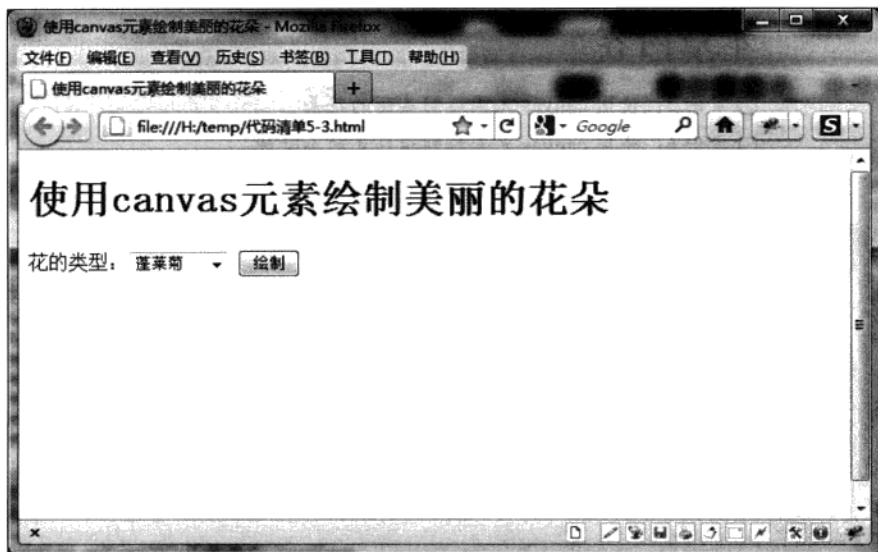


图 3-1 页面打开时的效果

不改变下拉框中的选项, 直接用鼠标单击绘制按钮后, 页面显示效果如图 3-2 所示。

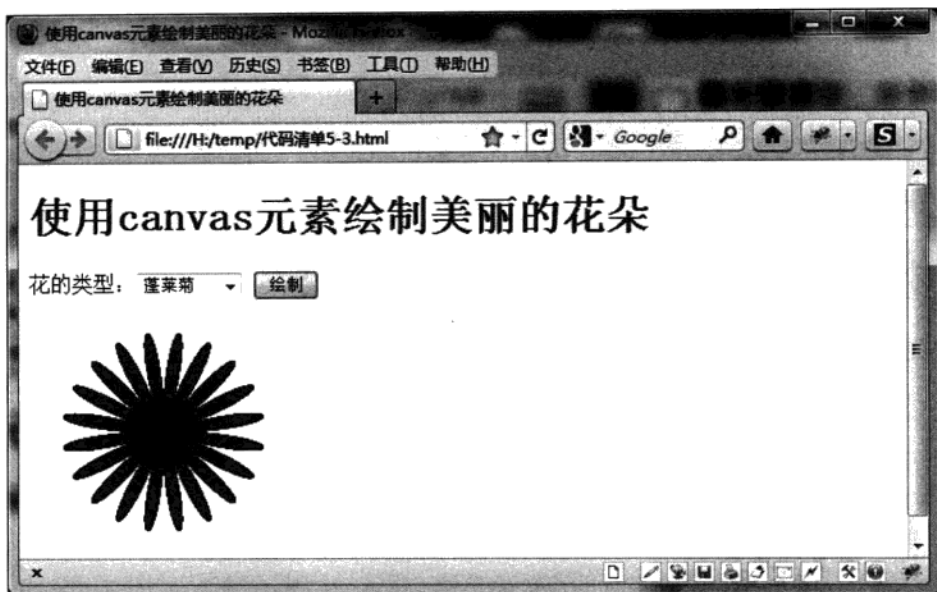


图 3-2 使用 canvas 元素绘制蓬莱菊

下拉框中除了“蓬莱菊”选项之外，还有“令箭荷花”与“大丽花”两个选项，选择“令箭荷花”选项，单击绘制按钮，页面显示效果如图 3-3 所示。



图 3-3 使用 canvas 元素绘制令箭荷花

选择“大丽花”选项，单击绘制按钮，页面显示效果如图 3-4 所示。



图 3-4 使用 canvas 元素绘制大丽花

### 3.1.3 案例知识点

在查看案例代码之前，首先对本例中用到的 Canvas API 中的几个属性与方法进行具体介绍。对 Canvas API 中全部属性与方法的全面介绍，可以查看笔者所著的《HTML 5 与 CSS 3 权威指南》一书，本章只结合案例具体介绍这些属性和方法的使用方法。

#### 1. 取得页面中的 canvas 元素

要使用 canvas 元素进行图形或图像的绘制，第一步工作就是在 JavaScript 脚本代码中使用如下语句取得页面中的 canvas 元素，并将其保存到某个变量中（通常将变量命名为 canvas 即可）。

```
var canvas=document.getElementById("canvas");
```

#### 2. 取得 canvas 元素的图形上下文对象

在大多数程序中进行图形绘制时，都需要使用图形上下文（graphics context）对象。图形上下文对象是一个封装了很多绘图功能的对象。在使用 canvas 元素进行图形绘制时，需要使用 canvas 对象的 getContext 方法来获得图形上下文对象，方法如下所示。

```
context=canvas.getContext('2d');
```

### 3. 向堆栈中保存当前绘制状态

在使用 canvas 元素进行图形或图像的绘制时，当需要对当前所绘制的图形或图像执行以下操作时，要先将当前绘制状态保存到堆栈中，然后进行相关操作，待操作完毕后再从堆栈中取出之前的绘制状态继续绘制。

需要保存绘制状态的操作为：

- ☐ 图像或图形变形
- ☐ 图像裁剪
- ☐ 改变图形上下文的以下属性：fillStyle,font,globalAlpha,globalCompositeOperation,lineCap,lineJoin, lineWidth,miterLimit,shadowBlur,shadowColor,shadowOffsetX,shadowOffsetY, strokeStyle,textAlign,textBaseline

在这些操作中，当执行变形操作时，因为需要改变当前绘制图形所用的坐标轴（将坐标轴平移、旋转或缩放），所以需要保存修改坐标轴之前的绘制状态，待变形操作执行完毕后，再从堆栈中取出之前的绘制状态（即恢复之前的坐标轴），然后继续进行绘制。当执行图像裁剪操作时，因为一旦设置好裁剪区域之后，后面绘制的所有图形都继续使用这个裁剪区域，所以需要恢复图像裁剪操作之前的绘制状态，以取消这个裁剪区域。当改变图形上下文属性时，可以先将当前绘制状态保存起来，然后修改相关属性，使用这些属性进行图形绘制之后，可以恢复之前保存的状态，即恢复之前使用的图形上下文的属性。

使用图形上下文对象的 save 方法来保存当前绘制状态，使用图形上下文对象的 restore 方法来恢复之前保存的绘制状态，代码如下所示。

```
context.save();
context.restore();
```

### 4. 将指定的矩形区域中的图形擦除

使用图形上下文对象的 clearRect 方法将利用 canvas 元素在某个指定区域中绘制的图形擦除，代码如下所示。

```
context.clearRect(x,y, width,height)
```

该方法使用四个参数：x 是指定的矩形区域起点的横坐标；y 是指定的矩形区域起点的纵坐标，坐标原点为 canvas 画布的左上角，width 是指定的矩形区域的长度，height 是指定的矩形区域的高度。本例每次更换下拉框中的选项后都要使用 canvas 元素重新进行花朵的绘制，因此在绘制前需要使用这个函数擦除之前利用 canvas 元素绘制好的图形。

### 5. 平移坐标轴

使用图形上下文对象的 translate 方法移动坐标轴原点。代码如下所示。

```
context.translate(x,y);
```

translate 方法有两个参数，x 表示将坐标轴原点向左移动多少个单位，默认单位为像素；y 表示将坐标轴原点向下移动多少个单位。

## 6. 创建路径

在进行复杂图形的绘制时，需要先创建该图形的绘制路径，然后以这个路径为基础进行图形的绘制。使用图形上下文对象的 `beginPath` 方法开始路径的创建工作，代码如下所示。

```
context.beginPath();
```

## 7. 设定绘图样式

在进行图形绘制的时候，首先要设定好绘图的样式（style），然后调用有关方法进行图形的绘制。所谓绘图的样式，主要是针对图形的颜色而言的，但是并不限于图形的颜色。

使用图形上下文对象的 `fillStyle` 属性来指定填充图形的样式，在该属性中填入填充的颜色值或其他填充样式，代码如下所示。

```
context.fillStyle="green";
```

使用图形上下文对象的 `strokeStyle` 属性来指定图形边框的样式，在该属性中填入图形边框的颜色值或其他边框样式，代码如下所示。

```
context.strokeStyle="black";
```

## 8. 绘制直线

使用图形上下文对象的 `lineTo` 方法绘制一条直线，代码如下所示。

```
lineTo(x,y)
```

该方法有两个参数，`x` 表示直线终点的横坐标，`y` 表示直线终点的纵坐标。使用该方法绘制完直线后，光标自动移动到 `lineTo` 方法的参数所指定的直线终点。

注意，该方法只是指定一条绘制直线的路径，绘制该直线的处理是依靠 `fill` 方法（填充图形）或 `stroke` 方法（绘制图形边框）执行的。

## 9. 填充图形

使用图形上下文对象的 `fill` 方法填充图形（填充样式依靠 `fillStyle` 属性来指定），代码如下所示。

```
context.fill();
```

## 10. 绘制图形边框

使用图形上下文对象的 `stroke` 方法绘制图形边框（边框样式依靠 `strokeStyle` 属性来指定），代码如下所示。

```
context.stroke();
```

## 11. 关闭路径

路径创建完成后，使用图形上下文对象的 `closePath` 方法将路径关闭，代码如下所示。

```
context.closePath();
```

将路径关闭后，路径的创建工作就完成了。

3.1.4 代码剖析

接下来具体分析一下该案例的整个实现过程及代码。

1. HTML 页面代码

案例页面所使用到的元素及其说明如表 3-1 中所示。

表 3-1 案例页面中所使用的元素及其说明

元素名称	元素类型	显示文字	说明
标题文字	h1	使用 canvas 元素绘制美丽的花朵	
花类型选择框的标题文字	(直接将文字书写在页面中)	花的类型:	
花类型选择框	select		下拉框中内容为: 蓬莱菊 令箭荷花 大丽花
绘制按钮	input type="button"	绘制	选择花的类型并且用鼠标单击绘制按钮时，案例程序将把所选择的花朵绘制在页面上
绘制花朵所用的画布	canvas		在该 canvas 元素中进行花朵的绘制与显示

案例页面的 HTML5 页面代码如代码清单 3-1 所示。

代码清单 3-1 案例页面的 HTML5 页面代码

```
<!DOCTYPE html>
<meta charset="UTF-8">
<title> 使用 canvas 元素绘制美丽的花朵 </title>
<script type="text/javascript">
... 脚本代码稍后介绍 ...
</script>
<h1> 使用 canvas 元素绘制美丽的花朵 </h1>
花的类型:
<select id="drawType">
    <option value="1"> 蓬莱菊 </option>
    <option value="2"> 令箭荷花 </option>
    <option value="3"> 大丽花 </option>
</select>
<input type="button" id="btn" value=" 绘制 " onclick="btn_onclick()" /><br/>
<canvas id="canvas" width="200px" height="200px"></canvas>
```



## 2. JavaScript 脚本代码

接下来，在代码清单 3-2 中看一下该案例完整的 JavaScript 脚本代码，之后对这个脚本代码进行详细分析。

代码清单 3-2 案例的完整 JavaScript 脚本代码（左边的数字表示行号）

---

```

<script type="text/javascript">
1   var context;
2   var A,n;
3   function btn_onclick()
4   {
5       var width;
6       var Height;
7       var canvas;
8       var Xo,Yo;
9       var k;
10      canvas=document.getElementById("canvas");
11      width=canvas.width;
12      height=canvas.height;
13      context=canvas.getContext('2d');
14      Xo=width/2;
15      Yo=height/2;
16      k=parseInt(document.getElementById("drawType").value);
17      if(k==2)
18          A=Yo*0.25;
19      else
20          A=Yo*0.75;
21      context.save();
22      context.clearRect(0,0,width,height);
23      context.translate(Xo,Yo);
24      context.beginPath();
25      for(var B=0;B<=6.28;B=B+0.01)
26      {
27          draw(B);
28      }
29      context.closePath();
30      context.restore();
31  }
32  function draw(B)
33  {
34      var n=10;
35      switch(parseInt(document.getElementById("drawType").value))
36      {
37          case 3:
38              r=A*Math.sin(n*B)*Math.exp(-B/(20));
39              break;
40          case 2:
41              r=A*(Math.sin(n*B)+3*Math.sin(3*n*B));
42              break;

```

```

43         case 1:
44             r=A*Math.sin(n*B);
45         }
46
47         x=r*Math.cos(B);
48         y=r*Math.sin(B);
49
50         context.fillStyle="green";
51         context.strokeStyle="black";
52         context.lineTo(-x,-y);
53         context.fill();
54         context.stroke();
55     }
</script>

```

在这段脚本代码的开始两行中，定义了三个全局变量——context、A 与 n。其中 context 变量代表画面中 canvas 元素的图形上下文对象，A 变量被用来控制花朵的半径长度（半径  $r=A\sin(nB)$ ），n 变量被用来控制所绘制的花朵中花瓣的数量。在本案例中，当绘制令箭荷花的时候，设置 A 等于 canvas 元素宽度（canvas 元素的宽度与高度都为 200px）的 1/8，当绘制蓬莱菊与大丽花的时候，设置 A 等于 canvas 元素宽度的 3/8，然后通过半径  $r=A\sin(nB)$  方程式来计算半径长度。另外，在本案例中，n 变量将被赋值为 10，表示绘制 20 片花瓣。

#### □ btn\_onclick 函数

在脚本代码中，从第 3 行开始到第 31 行结束为 btn\_onclick 函数的代码，当用户用鼠标单击绘制按钮时将调用该函数在 canvas 元素中进行花朵的绘制。

从第 5 行开始到第 9 行结束的代码，定义了本函数中使用的变量。其中变量 canvas 代表案例页面中的 canvas 元素，变量 width 与 height 表示 canvas 元素的宽度与高度，Xo 为 canvas 元素的宽度值的一半，Yo 为 canvas 元素的高度值的一半，由这两个变量所构成的坐标点即为 canvas 元素的中心点。变量 k 代表用户在花类型选择框中所选项的 value 值，绘制蓬莱菊时该值为 1，绘制令箭荷花时该值为 2，绘制大丽花时该值为 3。

在第 10 行开始到第 20 行结束的代码中，分别取得了页面中的 canvas 元素并保存在变量 canvas 中，取得 canvas 元素的宽度与高度并保存在变量 width 与变量 height 中，取得 canvas 元素的图形上下文对象并保存在变量 context 中，取得 canvas 元素的宽度与高度的一半并保存在变量 Xo 和变量 Yo 中，取得用户在花类型选择框中所选项的 value 值并保存在变量 k 中，同时根据所需绘制花的类型设置方程式  $r=A\sin(nB)$  中参数 A 的参数值。

代码第 21 行将图形上下文对象的当前绘制状态进行保存，因为在绘制花朵时需要使用图形上下文对象的 translate 方法平移坐标原点，所以在平移前首先保存绘制状态，在绘制花朵结束后再将绘制状态恢复。

在代码第 22 行中，将 canvas 元素中已经绘制的图形擦除，因为用户可能在 canvas 元素

中绘制了一朵花后重新选择花的类型，然后再次单击绘制按钮重新绘制花朵，这时需要将之前绘制的花朵擦除。在代码第 23 行中，将绘制图形的坐标原点移动到 canvas 元素中央，这样才可能以 canvas 元素的中心点为花朵的中心点进行花朵绘制。

从第 24 行开始到第 30 行结束的代码先创建图形路径，使方程式  $A\sin(nB)$  中的另一个参数  $B$  在从 0 开始到  $2\pi$  结束的范围内进行变动，每次将  $B$  增加 0.01，并在变动过程中循环调用 draw 函数进行花朵绘制。

最后，第 29 行代码关闭路径，第 30 行代码恢复之前保存的图形上下文对象的绘制状态。

#### □ draw 函数

在脚本代码中，从第 32 行开始到第 55 行结束的代码为 draw 函数。当用户用鼠标单击绘制按钮后，该函数接受一个参数  $B$ ，在 btn\_onclick 函数中使方程式  $A\sin(nB)$  中的另一个参数  $B$  在从 0 开始到  $2\pi$  结束的范围内进行变动，在这个变动过程中，会循环调用 draw 函数进行花朵绘制，并把每一个处于变动状态的参数  $B$  的数值作为参数传入到 draw 函数中。

在 draw 函数的开头处，即第 34 行代码处，把方程式  $A\sin(nB)$  中的  $n$  参数的数值设定为 10，表示绘制 20 片花瓣。

接下来，在第 35 行到第 45 行代码中，根据用户所选择的花的类型，使用不同的方程式来计算花朵半径的长度。在第 47 行与第 48 行代码中，将  $r*\text{Math.cos}(B)$  计算式与  $r*\text{Math.sin}(B)$  计算式的计算结果分别保存在变量  $x$  与变量  $y$  中。

在第 50 行到第 54 行代码中，在当前坐标点与  $(-x, -y)$  坐标点之间绘制一条直线，同时将当前坐标点移动到  $(-x, -y)$  坐标点处，绘制直线时的填充颜色为绿色，边框颜色为黑色。这样，通过不断变动参数  $B$  值来绘制直线，最终实现了花朵的绘制。

## 3.2 案例 6：使用 canvas 元素绘制指针式动画时钟

### 3.2.1 案例概述

接下来介绍一个使用 canvas 元素绘制图形动画的案例，该案例将绘制一个时钟，该时钟可以真实反映运行该案例程序的计算机的系统时间，并且每隔 1 秒钟该时钟的秒针会自动更新，使页面中的时钟始终与计算机的时钟保持同步。

### 3.2.2 页面显示效果

接下来看一下案例页面在浏览器中的显示效果，如图 3-5 所示。



图 3-5 使用 canvas 元素绘制的时钟

该时钟具有动画功能，即时钟的指针始终指向运行该页面的计算机的当前时间，并且随着时间的推移自动调整指针，使其始终指向当前时间，时钟下面的数字也始终显示当前时间。

### 3.2.3 案例知识点

接下来介绍这个案例页面中所用到的 Canvas API 的有关属性与方法。

在该页面中，除了使用前面所介绍的 Canvas API 的属性与方法之外，还使用了以下属性与方法。

#### 1. 创建圆形与弧形路径

在绘制时钟的圆形罗盘、时钟中心和弧形指针（看上去是直线指针，实际上略带弧度）的时候，都需要创建一个圆形或弧形路径。创建圆形或弧形路径均使用同一个方法，代码如下所示。

```
context.arc(x, y, radius, startAngle, endAngle, anticlockwise)
```

该方法有 6 个参数，x 为圆形的起点横坐标，y 为圆形的起点纵坐标，radius 为圆形半径，startAngle 为开始弧度，endAngle 为结束弧度，anticlockwise 为是否按顺时针方向进行绘制。

在 Canvas API 中，绘制半径与弧形时指定的参数为开始弧度与结束弧度，如果习惯使用角度，则使用如下方法将角度转换为弧度。

```
var radians=degrees*Math.PI/180
```

其中 `Math.PI` 表示角度为 180 度, `Math.PI*2` 表示角度为 360 度。

## 2. 指定所绘文字的字体属性

可以直接使用 `canvas` 元素的图形上下文对象的 `font` 属性来指定在 `canvas` 元素中绘制文字时所使用的字体属性。本例使用如下代码来指定所绘制的时钟上表示小时的文字与时钟下部表示具体时间的文字的字体属性。

```
context.font= 'bold 16px 宋体';
```

## 3. 指定绘制直线式的线宽

在本案例中, 需要指定指针的宽度。在 `Canvas API` 中, 可以直接使用图形上下文对象的 `lineWidth` 属性来指定线宽, 代码如下所示。

```
context.lineWidth=3
```

## 4. 得到文字的宽度

在绘制时钟上表示小时的文字时, 需要首先计算出所需文字的坐标, 在计算坐标的时候, 需要先得到文字的宽度。

在 `Canvas API` 中, 利用 `canvas` 元素的图形上下文对象的 `measureText` 方法来得到某个指定字符串在使用了指定字体大小后文字的宽度, 代码如下所示。

```
metrics=context.measureText(text);
```

`measureText` 方法接受一个参数 `text`, 该参数为需要指定的字符串, 该方法返回一个 `TextMetrics` 对象。`TextMetrics` 对象的 `width` 属性表示使用当前指定的字体后在 `text` 参数中指定字符串的总文字宽度。

## 5. 绘制文字

在 `Canvas API` 中, 使用 `canvas` 元素的图形上下文对象的 `fillText` 方法或 `strokeText` 方法来绘制文字。其中 `fillText` 方法以填充方式绘制字符串, 代码如下所示。

```
void fillText(text, x, y, [maxWidth]);
```

该方法接受 4 个参数, 第一个参数 `text` 表示要绘制的文字, 第二个参数 `x` 表示绘制文字的起点横坐标, 第三个参数 `y` 表示绘制文字的起点纵坐标, 第四个参数 `maxWidth` 为可选参数, 表示显示文字时的最大宽度, 可以防止文字溢出。

`strokeText` 方法以轮廓方式绘制字符串, 该方法的定义如下所示。

```
void stroke Text(text, x, y, [maxWidth]);
```

该方法的参数部分的解释与 `fillText` 方法的参数解释相同。

### 3.2.4 代码剖析

接下来具体分析一下该案例的整个实现过程及实现代码。

## 1. HTML 页面代码与样式代码

该案例的页面比较简单，只放置了两个 div 元素，一个 div 元素中放置了一个 h1 元素，在 h1 元素中显示标题文字“使用 canvas 元素绘制指针式动画时钟”，在另一个 div 元素中放置了一个用来绘制时钟的 canvas 元素。同时使用样式代码对 div 元素及 canvas 元素进行样式的指定。该案例的 HTML 页面代码与样式代码如代码清单 3-3 所示。

代码清单 3-3 案例的 HTML 页面代码与样式代码

---

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title> 使用 canvas 元素绘制指针式动画时钟 </title>
<script type="text/javascript">
.....脚本代码稍后介绍.....
</script>
<style>
div{
    display: -moz-box;
    display: -webkit-box;
    -moz-box-pack: center;
    -webkit-box-pack: center;
    width:100%
}
canvas{
    background-color:white;
}
</style>
</head>
<body onload="window_onload()">
<div><h1> 使用 canvas 元素绘制指针式动画时钟 </h1></div>
<div><canvas id="canvas" width="200px" height="200px"></canvas></div>
</body>
</html>
```

---

## 2. JavaScript 脚本代码

接下来，通过代码清单 3-4 重点介绍该案例的完整的 JavaScript 脚本代码，随后将对这个脚本代码进行详细分析。

代码清单 3-4 案例的完整的 JavaScript 脚本代码

---

```
<script type="text/javascript">
1   var canvas;
2   var context;
3   function window_onload()
4   {
5       canvas=document.getElementById("canvas");
6       context=canvas.getContext('2d');
```

---

```

7         setInterval("draw()",1000);
8     }
9     function draw()
10    {
11        var radius=Math.min(canvas.width / 2, canvas.height / 2) -25;
12        var centerx=canvas.width/2;
13        var centery=canvas.height/2;
14        context.clearRect(0,0,canvas.width,canvas.height);
15        context.save();
16        context.fillStyle = '#efefef';
17        context.strokeStyle = '#c0c0c0';
18        context.beginPath();
19        context.arc(centerx,centery,radius, 0,Math.PI*2, 0);
20        context.fill();
21        context.stroke();
22        context.closePath();
23        context.restore();
24        var r = radius - 10;
25        context.font= 'bold 16px 宋体';
26        Drawtext('1', centerx + (0.5 * r), centery - (0.88 * r));
27        Drawtext('2', centerx + (0.866 * r), centery - (0.5 * r));
28        Drawtext('3', centerx + radius - 10,centery);
29        Drawtext('4', centerx + (0.866 * r), centery + (0.5 * r));
30        Drawtext('5', centerx + (0.5 * r), centery + (0.866 * r));
31        Drawtext('6', centerx, centery + r);
32        Drawtext('7', centerx - (0.5 * r), centery + (0.866 * r));
33        Drawtext('8', centerx - (0.866 * r), centery + (0.49 * r));
34        Drawtext('9', centerx - radius + 10, centery);
35        Drawtext('10',centerx - (0.866 * r),centery-(0.50 * r));
36        Drawtext('11', centerx - (0.51 * r), centery - (0.88 * r));
37        Drawtext('12', centerx, 35);
38
39        var date=new Date();
40        var h = date.getHours();
41        var m = date.getMinutes();
42        var s=date.getSeconds();
43        var a = ((h/12) *Math.PI*2) - 1.57 + ((m / 60) * 0.524);
44
45        context.save();
46        context.fillStyle='black';
47        context.beginPath();
48        context.arc(centerx,centery,3,0,Math.PI * 2, 0);
49        context.closePath();
50        context.fill();
51
52        context.lineWidth=3;
53        context.fillStyle='darkgray';
54        context.strokeStyle='darkgray';
55        context.beginPath();
56        context.arc(centerx,centery,radius - 55, a + 0.01, a, 1);

```

```

57     context.lineTo(centerx,centery);
58     context.arc(centerx,centery,radius - 40, ((m/60) * 6.27) - 1.57,
    ((m/60) * 6.28) - 1.57, 0);
59     context.lineTo(canvas.width / 2, canvas.height / 2);
60     context.arc(centerx,centery,radius - 30, ((s/60) * 6.27) - 1.57,
    ((s/60) * 6.28) - 1.57, 0);
61     context.lineTo(centerx,centery);
62     context.closePath();
63     context.fill();
64     context.stroke();
65     context.restore();
66
67     var hours    = String(h);
68     var minutes  = String(m);
69     var seconds  = String(s);
70
71     if (hours.length == 1) h   = '0' + h;
72     if (minutes.length == 1) m = '0' + m;
73     if (seconds.length == 1) s = '0' + s;
74     var str =h + ':' + m + ':' + s;
75     Drawtext(str, centerx, centery + radius + 12);
76 }
77 function Drawtext(text, x, y)
78 {
79     context.save();
80     x -= (context.measureText(text).width / 2);
81     y +=9;
82     context.beginPath();
83     context.translate(x, y);
84     context.fillText(text,0,0);
85     context.restore();
86 }
</script>

```

这段脚本代码的开始两行定义了案例脚本中使用到的两个全局变量，其中 canvas 变量代表页面中的 canvas 元素，context 变量代表该 canvas 元素的图形上下文对象。

从第 3 行开始到第 8 行结束的代码为 window\_onload 函数，打开页面时将调用该函数，该函数内容十分简单，分别取得页面中的 canvas 元素并保存在 canvas 变量中，取得该 canvas 元素的图形上下文对象并保存在 context 变量中，然后指定每隔 1 秒钟执行一次 draw 函数，在 draw 函数中进行时钟重绘。

从第 9 行开始到第 76 行结束的代码为 draw 函数，该函数执行每隔 1 秒钟的时钟重绘。

在第 11 行中，指定时钟罗盘的半径，当 canvas 元素的宽度小于高度时，罗盘半径为 canvas 元素的宽度 -25，当 canvas 元素的高度小于宽度时，罗盘半径为 canvas 元素的高度 -25。

第 12 行与第 13 行代码分别将 canvas 元素中心点的横坐标与纵坐标分别保存在变量



centerx 与变量 centery 中。

第 14 行代码执行擦除上一秒钟所绘制的时钟图形。

第 15 行代码执行将当前图形上下文对象的绘制状态保存起来。因为接下来绘制时钟罗盘时需要修改绘制图形时所使用的填充样式与边框样式（通过修改图形上下文对象的 fillStyle 属性与 strokeStyle 属性），所以在修改前先将当前图形上下文对象的绘制状态保存起来，以便在绘制时钟罗盘完毕之后再将图形上下文对象的绘制状态恢复以继续使用。

从第 16 行开始到第 23 行结束的代码为绘制时钟圆盘的代码。在这段代码中，分别指定时钟圆盘背景色为浅灰色，边框为深灰色，然后创建路径，用浅灰色填充罗盘，用深灰色绘制时钟圆盘边框，最后关闭路径并恢复图形上下文对象的绘制状态。

从第 24 行开始到第 37 行结束的代码为时钟上表示小时的文字的代码。在这段代码中，先将变量 r 的值设定为时钟半径的长度减去 10，然后将变量 r 作为绘制时钟上表示小时的文字时所使用的半径。因为要将文字绘制在时钟内部，所以绘制小时文字时所使用的半径要比时钟半径小一些。然后计算从 1 到 12 这 12 个小时的小时文字在时钟上的横坐标与纵坐标，并调用 Drawtext 函数绘制这 12 个小时的文字。

从第 39 行开始到第 65 行结束的代码为绘制时钟指针的代码。

从第 39 行开始到第 43 行结束的代码，分别取得运行本案例程序的计算机的当前时间的时、分、秒，并计算时针的绘制弧度，分别保存在变量 h、m、s 与 a 中。

从第 45 行开始到第 50 行结束的代码为绘制位于时钟中心部一个实心小圆圈的代码。在这段代码中，仍然先将当前图形上下文对象的绘制状态保存起来。因为在接下来的代码中绘制时钟中心圆圈时仍然需要修改绘制图形时所使用的填充样式与边框样式。在保存了当前图形上下文对象的绘制状态之后，使用黑色填充位于时钟中心部的实心小圆圈。

从第 52 行开始到第 65 行结束的代码分别绘制了时钟的时针、分针与秒针。在这段代码中，首先指定时钟的时针、分针与秒针全部为深灰色，然后开始创建路径，分别绘制时针、分针与秒针，最后在第 65 行代码中将之前保存的图形上下文对象从绘制状态恢复出来。

从 67 行开始到第 75 行结束的代码为绘制位于时钟下部的当前时间字符串的代码。当前时间字符串的格式为“HH:MM:SS”，67 行到 73 行的代码判断之前保存在变量 h、m、s 中的小时文字、分钟文字与秒钟文字的长度是否为两位，如果只有一位，则在该文字前补一个 0，然后将小时文字、分钟文字与秒钟文字连接成一个字符串，文字与文字之间用冒号分隔，最后用 DrawText 函数将其绘制在页面上。

从第 77 行开始到第 86 行结束的代码为 DrawText 函数，该函数的功能是在 canvas 元素中绘制文字。该函数接受 3 个参数，分别为要绘制的文字，该文字的横坐标与纵坐标。

第 79 行代码先保存当前图形上下文对象的绘制状态，因为在绘制文字的时候需要将坐标原点移动到函数参数所指定的坐标点处。接下来，使用图形上下文对象的 measureText 方法计算指定绘制文字在使用了当前图形上下文对象中所指定的字体属性后的文字宽度，然后将参数中指定的 x 坐标值减去该文字宽度的一半，使指定的 x 坐标值成为绘制文字的中心点处的横坐标，同时将指定的 y 坐标值向下偏移 9 个单位，使时钟上的小时文字不要与时钟边

框挤在一起,时钟下部的当前时间字符串的文字不要紧挨着时钟。接下来,将坐标原点移动到由修改后的 x 坐标值与 y 坐标值所指定的 (x,y) 坐标点处进行文字绘制,绘制完毕后对保存的图形上下文对象的绘制状态进行恢复。

### 3.3 案例 7: 使用 canvas 元素制作简单小游戏

#### 3.3.1 案例概述

接下来看一个使用 canvas 元素制作的简单小游戏。该案例为一个用户捕捉处于弹跳状态的小球的小游戏。案例中有一个 canvas 元素,该 canvas 元素用来模拟一个游戏桌面。单击案例页面中的开始游戏按钮后桌面上出现一个处于弹跳状态的小球,用户单击该小球,如果击中小球则弹出提示信息,提示用户游戏成功并结束。

#### 3.3.2 页面显示效果

首先看一下案例页面在浏览器中打开时的显示效果,如图 3-6 所示。



图 3-6 案例程序在浏览器中打开时的效果

单击开始游戏按钮后,画面中出现一个矩形桌面,桌面中有一个蓝色小球,该小球在该边框中处于不断游走弹跳(小球位置不断变换)的状态,同时开始按钮也变为无效状态,如图 3-7 所示。

用户单击该小球,击中小球后将弹出提示信息框,提示用户获胜,如图 3-8 所示。

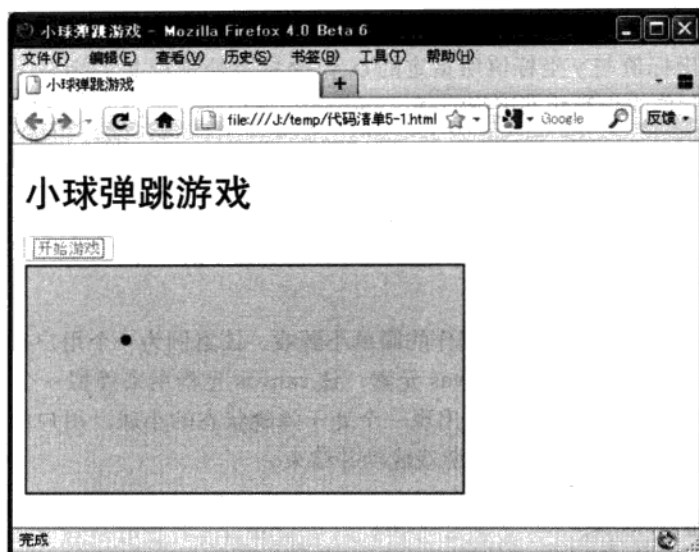


图 3-7 单击开始游戏按钮后画面中出现一个不断游走的小球



图 3-8 用户击中小球后游戏结束

单击提示信息框中的确定按钮后游戏结束，小球处于静止状态，开始游戏按钮恢复为有效状态，如图 3-9 所示。



图 3-9 单击信息提示框中的确定按钮后开始游戏按钮恢复为有效状态

### 3.3.3 案例知识点

接下来看一下这个案例中所使用的 Canvas API 中的一些属性与方法。除了前面已经介绍过的属性与方法外，为了使案例中利用 canvas 元素绘制的彩色桌面尽量具有立体效果，分别使用了填充矩形与绘制矩形边框两种方法，其中填充矩形的使用方法如下所示。

```
context.fillRect(x,y,width,height)
```

fillRect 方法使用 4 个参数，其中 x 是指矩形起点的横坐标，y 是指矩形起点的纵坐标，坐标原点为 canvas 画布的左上角，width 是指矩形的长度，height 是指矩形的高度。通过这四个参数，矩形的大小也同时被确定了。

绘制矩形边框的方法如下所示。

```
context.strokeRect(x,y,width,height)
```

strokeRect 方法中 4 个参数的含义与 fillRect 方法中 4 个参数的含义完全相同。

本程序绘制了一个填充色为淡绿色的桌面，并且绘制黑色桌面外框，为了能完全显示左边框与上边框，所绘制的矩形的坐标起点为 (3, 3)，程序代码如下所示。

```
context.fillStyle="lightgreen";
context.strokeStyle="black";
context.lineWidth=3;
context.fillRect(3,3,width-5,height-5);
context.strokeRect(3,3,width-5,height-5);
```

### 3.3.4 代码剖析

最后来看一下该案例程序的完整代码，如代码清单 3-5 所示，稍后将对案例代码进行详细分析。

代码清单 3-5 小球弹跳游戏的完整代码

---

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title> 小球弹跳游戏 </title>
<script type="text/javascript">
var BallX,BallY;
var AddX,AddY;
var width,height;
var canvas;
var context;
var functionId;

function btnBegin_onclick()
{
    canvas=document.getElementById("canvas");
    width=canvas.width;
    height=canvas.height;
    context=canvas.getContext('2d');

    BallX=parseInt(Math.random()*canvas.width);
    BallY=parseInt(Math.random()*canvas.height);
    AddX=-5;
    AddY=-5;
    draw();

    document.getElementById("btnBegin").disabled="disabled";
    functionId=setInterval("draw()",100);
}

function draw()
{
    context.clearRect(0,0,width,height);
    context.save();
    context.fillStyle="lightgreen";
    context.strokeStyle="black";
    context.lineWidth=3;
    context.fillRect(3,3,width-5,height-5);
    context.strokeRect(3,3,width-5,height-5);
    context.beginPath();
    context.fillStyle="blue";
    context.arc(BallX,BallY, 5,0,Math.PI * 2,false);
    BallX+=AddX;
    BallY+=AddY;
```

```

    if (BallX<5)
    {
        BallX=5;
        AddX=-AddX;
    }
    else if (BallX>width-5)
    {
        BallX=width-5;
        AddX=-AddX;
    }
    if (BallY<5)
    {
        BallY=5;
        AddY=-AddY;
    }
    else if (BallY>height-5)
    {
        BallY=height-5;
        AddY=-AddY;
    }
    context.closePath();
    context.fill();
    context.restore();
}
function canvas_mouseup(ev)
{
    var differenceX;
    var differenceY;

    differenceX=ev.pageX-document.getElementById("canvas").offsetLeft
    -BallX;

    differenceY=ev.pageY-document.getElementById("canvas").offsetTop
    -BallY;
    if (-5<=differenceX&&differenceX<=5)
        if (-5<=differenceY&&differenceY<=5)
        {
            alert("恭喜您获胜! 游戏结束");
            clearInterval(functionId);

            document.getElementById("btnBegin").disabled="";
        }
}
function window_onload()
{
    document.getElementById("canvas").onmouseup=canvas_mouseup;
}
</script>
</head>
<body onload="window_onload()">

```

```

<h1> 小球弹跳游戏 </h1>
<input type="button" id="btnBegin" value=" 开始游戏 "
onclick="btnBegin_onclick()" /><br/>
<canvas id="canvas" width=400px height=200px></canvas>
</body>
</html>

```

案例页面的 HTML 代码部分比较简单，首先使用 h1 元素显示页面标题文字“小球弹跳游戏”，然后利用 button 类型的 input 元素来设计开始游戏按钮，利用一个 canvas 元素来模拟小球弹跳游戏的桌面。

接下来介绍一下案例页面的 JavaScript 脚本代码部分。

在此部分脚本代码中，首先定义了本案例中使用的几个全局变量，变量含义如下。

- ❑ BallX 与 BallY：小球中心点在 canvas 元素中的横坐标与纵坐标。
- ❑ AddX 与 AddY：在运动过程中小球每次移动时的横向移动距离与纵向移动距离（以像素为单位）。
- ❑ width 与 height：canvas 元素的宽度与高度。
- ❑ canvas：页面中的 canvas 元素。
- ❑ context：canvas 元素的图形上下文对象。
- ❑ functionId：用来停止小球弹跳动画函数的整型变量。小球弹跳动画开始与结束的代码如下所示。

```

// 定义每隔 100 毫秒调用 draw 函数来进行小球的重绘，从而实现小球弹跳动画效果
functionId=setInterval("draw()",100);
// 停止小球动画
clearInterval(functionId);

```

在定义全局变量之后，定义了几个本案例程序中所用到的函数。

#### ❑ btnBegin\_onclick 函数

单击开始按钮时，将调用 btnBegin\_onclick 函数。

该函数先获取页面中的 canvas 元素并将其赋值给 canvas 变量，获取 canvas 元素的宽度与高度并赋值给 width 变量与 height 变量中，获取 canvas 元素的图形上下文对象并赋值给 context 变量。

接下来将小球中心点的横坐标值 (BallX) 与纵坐标值 (BallY) 设定为 canvas 元素中的一个随机坐标点的横坐标值与纵坐标值。设定在小球弹跳动画中小球每次移动时的横向移动距离 (AddX) 与纵向移动距离 (AddY) 均为 5 个像素，变量值为 -5 表示小球向左移动与向上移动 5 个像素。

然后先调用一次 draw 函数绘制矩形桌面与桌面中的小球，绘制完毕后设置开始游戏按钮为无效状态，这样可以防止用户在游戏开始到游戏结束这一段时间内再次单击开始游戏按钮。最后，调用 JavaScript 中的 setInterval 函数来设定每隔 100 毫秒重新调用 draw 函数来重绘矩形桌面与小球以产生小球弹跳动画效果。

#### □ draw 函数

本案例通过 draw 函数来绘制 canvas 元素中的矩形桌面与小球。

在该函数中，首先调用图形上下文对象的 clearRect 函数以擦除上次在 canvas 元素中绘制的小球。然后调用图形上下文对象的 save 方法保存图形上下文对象的当前绘制状态，因为接下来要通过修改图形上下文对象的 fillStyle 属性与 strokeStyle 属性来改变绘制时的图形填充颜色与边框颜色。在函数结尾部分通过图形上下文对象的 restore 方法来恢复图形上下文对象的当前绘制状态。

接下来，设定矩形桌面的颜色为绿色，边框为黑色，边框宽度为 3 个像素，然后绘制矩形桌面。

绘制完矩形桌面后设定小球为蓝色，然后创建小球的绘制路径，小球的半径为 5。同时根据小球中心点的当前坐标与每次的移动距离设定下次绘制小球时小球中心点的横坐标值 (BallX) 与纵坐标值 (BallY)，如果下次小球中心点的横坐标值与纵坐标值小于 5，或者大于 canvas 元素的宽度 -5 或 canvas 元素的高度 -5，则表示下次绘制的小球将超出 canvas 元素的边框，这时需要将下次要绘制的小球的中心点的横坐标值或纵坐标值设定为 5 或 canvas 元素的宽度（或高度）-5，以使下次绘制的小球正好在 canvas 元素的内部，同时改变小球的移动方向（将变量 AddX 与变量 AddY 从正数变成负数，或从负数变成正数，正数表示向右或向下移动，负数表示向左或向上移动）。

创建完小球弹动路径并计算下次（隔 100 毫秒）要绘制的小球的中心点的横坐标值与纵坐标值后关闭路径，绘制小球，然后恢复 canvas 元素的图形上下文对象的绘制状态。

#### □ canvas\_mouseup 函数

当用户按下鼠标试图击中小球后松开鼠标时将调用该函数。

在该函数中，首先计算鼠标击中点的横坐标与小球中心点的横坐标值之间的距离并保存在变量 differenceX 中，计算鼠标击中点的纵坐标与小球中心点的纵坐标值之间的距离并保存在变量 differenceY 中。因为小球的半径为 5，所以，如果鼠标击中点的横坐标与小球中心点的横坐标值之间的距离及鼠标击中点的纵坐标与小球中心点的纵坐标值之间的距离均小于 5 则表示用户击中小球，这时弹出提示信息，提示用户击中小球，同时停止小球弹跳动画并设定开始游戏按钮为有效状态，以便让用户再次单击开始游戏按钮，进行下一次游戏。

#### □ window\_onload 函数

页面打开时将调用 window\_onload 函数。在该函数中对 canvas 元素添加对 mouseup 事件的监听，设定当用户按下鼠标试图击中小球后松开鼠标时将调用 canvas\_mouseup 函数。

### 3.4 案例 8：使用 canvas 元素绘制图像放大镜

使用 HTML 5 中的 canvas 元素以及 Canvas API，不仅可以绘制图形，还可以绘制图像、处理图像处理，以及制作各种之前必须使用 Flash 才能绘制的图像动画。

本节介绍使用 canvas 元素绘制图像放大镜的一个案例。



### 3.4.1 案例概述

在网页中，经常会由于显示空间不够或出于性能考虑等在绘制图片时采取不按原尺寸绘制全图，而将该图尺寸缩小，或重新制作小尺寸的缩略图，然后只显示缩小后的缩略图。如果用户想看清楚该图，就必须单击该图，然后在浏览器中重新打开一个新的窗口或标签才能查看原图。

本案例将利用 canvas 元素为缩略图添加一个放大镜，当用户将鼠标移动到该缩略图上时，会自动出现一个放大镜，将鼠标所在区域（本例中为鼠标的右下角 40 像素 × 40 像素的图像区域）进行放大显示（本例中为放大 4 倍，横向放大 2 倍，纵向放大 2 倍），并且随着鼠标的移动会自动更新放大镜中的放大区域，当用户将鼠标移动到图像之外时，该放大镜被自动隐藏。

### 3.4.2 页面显示效果

首先来看一下案例页面在浏览器中的显示效果，如图 3-10 所示。

当鼠标移动到图像上时，会自动在图像上出现一个放大镜，在放大镜中放大显示鼠标指针所在位置的局部区域的图像，并随着鼠标的移动自动更新放大区域，如图 3-11 所示。



图 3-10 实例页面在浏览器中打开时的显示效果



图 3-11 在图像上显示一个放大镜

当鼠标移出图像外时，该放大镜会被自动隐藏。

### 3.4.3 案例知识点

接下来看一下这个案例程序所使用的 Canvas API 中的一些属性与方法，除了前面已经介绍过的属性与方法外，该程序主要使用了 Canvas API 中的绘制图像的方法。

在 Canvas API 中，使用 drawImage 方法绘制图像。该方法的定义如下所示。

```
context.drawImage(image,x,y);
context.drawImage(image,x,y,w,h);
context.drawImage(image,sx,sy,sw,sh,dx,dy,dw,dh);
```

第一种方法只使用三个参数，第一个参数 image 可以是一个 img 元素、一个 video 元素或者 JavaScript 中的一个 Image 对象，使用该参数实际代表的对象来装载图像。x 与 y 为绘制时该图像在画布中的起始坐标。

第二种方法中的前三个参数与第一种方法中的一样，w、h 是指绘制时的图像的宽度与高度。第一种方法中省略了这两个参数，所以绘制出来的图像与原图大小相同，而第二种方法可以对图像进行缩放。

第三种方法可以将某幅图像的全部或者局部区域复制到画布中的另一个位置上。该方法有 9 个参数，image 仍然代表被复制的图像文件，sx 与 sy 分别表示被复制的源图像所在的区域在画布中的起始横坐标与起始纵坐标，sw 与 sh 表示被复制区域的宽度与高度，dx 与 dy 表示复制后的目标图像在画布中的起始横坐标与起始纵坐标，dw 与 dh 表示复制后的目标图像的宽度与高度。该方法可以只复制图像的局部，只要将 sx 与 sy 设为局部区域的起始点坐标，将 sw 与 sh 设为局部区域的宽度与高度就可以了。该方法也可以对源图像进行缩放，只要将 dw 与 dh 设为缩放后的宽度与高度就可以了。

本案例同时使用了这三种方法中的第一种方法与第三种方法，使用第一种方法来绘制原图，代码如下所示。

```
context.drawImage(image,0,0);
```

使用第三种方法来将原图像中鼠标所指区域绘制到放大镜中，代码如下所示。

```
context.drawImage(image,x,y,drawWidth,drawHeight,0,0,drawWidth*2,
drawHeight*2);
```

### 3.4.4 代码剖析

接下来具体分析本案例的整个实现过程及实现代码。

#### 1. HTML 页面代码与样式代码

本案例页面的 HTML 页面代码部分的内容比较简单，利用 h1 元素显示页面标题文字“对图像使用放大镜”。然后使用两个 canvas 元素，在第一个 canvas 元素中绘制一幅图像，在第二个 canvas 元素中绘制放大镜。

在样式代码中，为了使绘制出来的放大镜为圆形放大镜，利用 CSS3 中的 border-radius 属性来绘制放大镜（canvas 元素）的圆角边框，并指定圆角半径为 40px。另外，为了让放大镜能够叠放在需要放大的原图像上面，指定显示原图像的 canvas 元素的 z-index 属性的属性值为 1，显示放大镜的 canvas 元素的 z-index 属性的属性值为 2。

案例页面的完整的 HTML 页面代码与样式代码如代码清单 3-6 所示。

代码清单 3-6 案例页面的完整的 HTML 页面代码与样式代码

---

```

<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title> 对图像使用放大镜 </title>
<script type="text/javascript">
... 脚本代码稍后介绍 ...
</script>
<style>
canvas{
    background-color:white;
    position:absolute;
}
canvas#canvas1{
    z-index:1;
}
canvas#canvas2{
    z-index:2;
    left:0px;
    top:0px;
    border:thin dashed black;
    border-radius: 40px;
    -moz-border-radius: 40px;
    -o-border-radius: 40px;
    -webkit-border-radius: 40px;
    display:none;
    position:relative;
}
</style>
</head>
<body onload="window_onload()">
<article>
<h1> 对图像使用放大镜 </h1>
<canvas id="canvas1" width="100px" height="130px"></canvas>
<canvas id="canvas2" width="80px" height="82px"></canvas>
</article>
</body>
</html>

```

---

## 2. JavaScript 脚本代码

代码清单 3-7 显示了本案例中完整的 JavaScript 脚本代码，后面将对这个脚本代码进行详细分析。

代码清单 3-7 案例的完整 JavaScript 脚本代码

---

```

<script type="text/javascript">
1   function window_onload()
2   {

```

---

```

3      var canvas1 = document.getElementById('canvas1');
4      context = canvas1.getContext('2d');
5      var image = new Image();
6      image.src = "tyl.jpg";
7      image.onload=function(){
8          context.drawImage(image,0,0);
9      }
10
11      canvas1.onmousemove=canvas1_onmouse_move;
12      canvas1.onmouseout=canvas1_onmouse_out;
13  }
14  function canvas1_onmouse_move(ev)
15  {
16      var canvas1,canvas2,context,image;
17      var x,y;
18      var drawWidth,drawHeight;
19      canvas1=document.getElementById("canvas1");
20      canvas2=document.getElementById("canvas2");
21      context = canvas2.getContext('2d');
22
23      canvas2.style.display="inline";
24      context.clearRect(0,0,canvas2.width,canvas2.height);
25
26      x=ev.pageX-canvas1.offsetLeft;
27      y=ev.pageY-canvas1.offsetTop;
28      canvas2.style.left=x+2+"px";
29      canvas2.style.top=y+2+"px";
30
31      if(x+40>canvas1.width)
32          drawWidth=canvas1.width-x;
33      else
34          drawWidth=40;
35
36      if(y+40>canvas1.height)
37          drawHeight=canvas1.height-y;
38      else
39          drawHeight=40;
40
41      var image = new Image();
42      image.src = "tyl.jpg";
43      context.drawImage(image,x,y,drawWidth,drawHeight,0,0,
44          drawWidth*2,drawHeight*2);
45  }
46  function canvas1_onmouse_out()
47  {
48      var canvas2=document.getElementById("canvas2");
49      canvas2.style.display="none";
50  }
51  }
52  </script>

```

---

本案例的脚本代码中共有三个函数——`window onload` 函数、`canvas1_onmouse_move` 函数与 `canvas1_onmouse_out` 函数。打开页面时将调用 `window onload` 函数，当鼠标指针在被放大的图像上移动时将调用 `canvas1_onmouse_move` 函数，当鼠标指针从被放大的图像上移出时将调用 `canvas1_onmouse_out` 函数。

#### □ `window onload` 函数

`window onload` 函数首先获取页面中用来绘制被放大图像的 `canvas` 元素（以下简称源图像 `canvas` 元素）以及该 `canvas` 元素的图形上下文对象，然后通过 JavaScript 的 `Image` 对象装载一幅图像，并将该图像绘制在 `canvas` 元素中，同时指定该 `canvas` 元素监听两个事件——`mouse_move` 事件与 `mouse_out` 事件，指定当鼠标指针在源图像 `canvas` 元素中移动时调用 `canvas1_onmouse_move` 函数，当鼠标指针从源图像 `canvas` 元素中移出时调用 `canvas1_onmouse_out` 函数。

#### □ `canvas1_onmouse_move` 函数

`canvas1_onmouse_move` 函数的开头三行指定了本函数中所使用到的变量，这些变量的含义如下。

- `canvas1`：源图像 `canvas` 元素。
- `canvas2`：用来绘制放大镜的 `canvas` 元素（以下简称放大镜 `canvas` 元素）。
- `context`：放大镜 `canvas` 元素的图形上下文对象。
- `image`：在放大镜中绘制的图像。
- `x` 与 `y`：被放大区域在源图像 `canvas` 元素中的起始点的横坐标值与纵坐标值。
- `drawWidth` 与 `drawHeight`：被放大区域的宽度与高度。

从第 19 行到第 21 行结束的代码分别获取页面中的源图像 `canvas` 元素并赋值给 `canvas1` 变量，获取页面中的放大镜 `canvas` 元素并赋值给 `canvas2` 变量，获取放大镜 `canvas` 元素的图形上下文对象并赋值给 `context` 变量。

第 23 行代码将放大镜 `canvas` 元素的 `display` 样式属性的属性值设定为 `inline`，使放大镜可以显示在源图像 `canvas` 元素中，或者说可以让放大镜显示在被放大的图像上面。

第 24 行代码擦除放大镜中之前绘制的图像，因为鼠标指针可能处于不断移动的状态中，所以移动鼠标指针的位置后就需要将移动前在放大镜中绘制的图像擦除。

从第 26 行到第 29 行的代码计算移动鼠标时鼠标指针在源图像 `canvas` 元素中的横坐标值与纵坐标值（`ev.pageX` 与 `ev.pageY` 表示鼠标指针在页面中的横坐标值与纵坐标值，`canvas1.offsetLeft` 与 `canvas1.offsetTop` 表示源图像 `canvas` 元素的左上角在页面中的横坐标值与纵坐标值），然后将横坐标值与纵坐标值均加 2，将其设定为放大镜 `canvas` 元素的左上角的横坐标值与纵坐标值。加 2 是为了避免鼠标指针被移动到放大镜上，因为一旦鼠标指针被移动到放大镜上，就不会再调用 `canvas1_onmouse_move` 函数来进行鼠标指针移动后在放大镜中根据鼠标指针所在位置来绘制放大图像的处理了。在计算出移动鼠标时鼠标指针在源图像 `canvas` 元素中的横坐标值与纵坐标值之后，将放大镜 `canvas` 元素左上角的横坐标值与纵坐标值设定为计算出来的横坐标值与纵坐标值。

从第 31 行到第 39 行的代码，计算源图像 canvas 元素中被放大区域的宽度与高度，默认被放大区域的宽度与高度均为 40，但是如果鼠标指针所在的横坐标值或纵坐标值加上 40 后超出源图像 canvas 元素的宽度或高度，指定被放大区域的宽度或高度为从鼠标指针所在点的横坐标值或纵坐标值到源图像 canvas 元素的宽度或高度之间的距离。

canvas1\_onmouse\_move 函数的最后三行使用 JavaScript 中的 Image 对象再一次装载要放大的图像，并将该图像放大绘制在放大镜 canvas 元素中。

#### □ canvas1\_onmouse\_out 函数

当鼠标指针从源图像 canvas 元素中移出去时将调用 canvas1\_onmouse\_out 函数。该函数中的代码比较简单，所进行的处理为将放大镜元素隐藏显示。

## 3.5 案例 9：用动画的形式装载图像

### 3.5.1 案例概述

在 HTML 5 之前，在网页上装载一幅图像时，只能使用 img 元素，以静态的形式来装载图像。如果使用 canvas 元素，则可以用动画的形式来装载图像，或者说不是一下子打开图像，而是以动画的形式打开图像。本案例的页面中有几个按钮，单击这些按钮可以以各种不同的动画形式来装载一幅图像。

### 3.5.2 页面显示效果

本案例页面在浏览器中打开时的显示效果如图 3-12 所示。

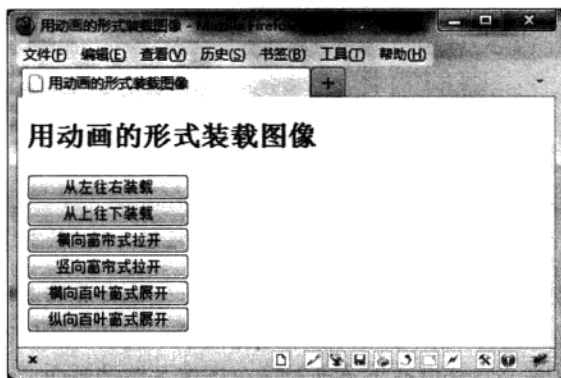


图 3-12 案例页面在浏览器中打开时的显示效果

单击“从左往右装载”按钮后，按钮右侧的 canvas 元素开始装载图像，装载方式为从左往右动态装载，如图 3-13 所示，直到图像完全装载完毕。在装载的同时，为了防止用户再次单击按钮而导致同时对同一个 canvas 元素执行多个图像装载动画而产生问题，因此把所有按

钮设为无效状态。

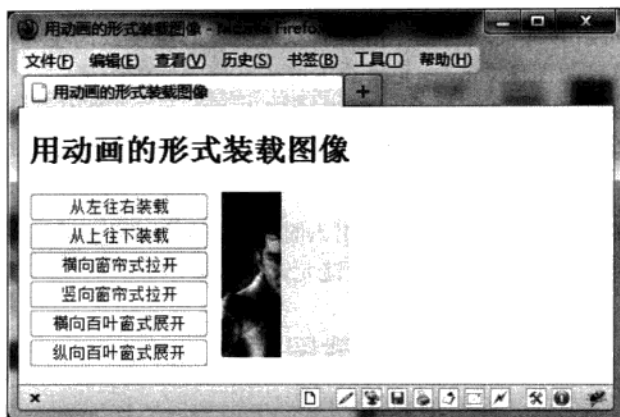


图 3-13 从左往右装载图像

图像装载完毕后所有按钮变为有效状态，如图 3-14 所示，可以再次单击按钮执行图像装载动画。



图 3-14 图像装载完毕后所有按钮恢复为有效状态

单击“从上往下装载”按钮后，按钮右侧的 canvas 元素开始装载图像，装载方式为从上往下动态装载，如图 3-15 所示，直到图像完全装载完毕。在装载的同时，为了防止用户再次单击按钮而导致同时对同一个 canvas 元素执行多个图像装载动画而产生问题，因此把所有按钮设为无效状态。图像全部装载完毕后，所有按钮恢复为有效状态。

单击“横向窗帘式拉开”按钮后，按钮右侧的 canvas 元素开始装载图像，装载方式为首先显示图像中央部分，然后以向左右两边逐渐拉开的方式动态装载图像，如图 3-16 所示，直到图像完全装载完毕。在装载的同时，把所有按钮设为无效状态。图像全部装载完毕后，所

有按钮恢复为有效状态。

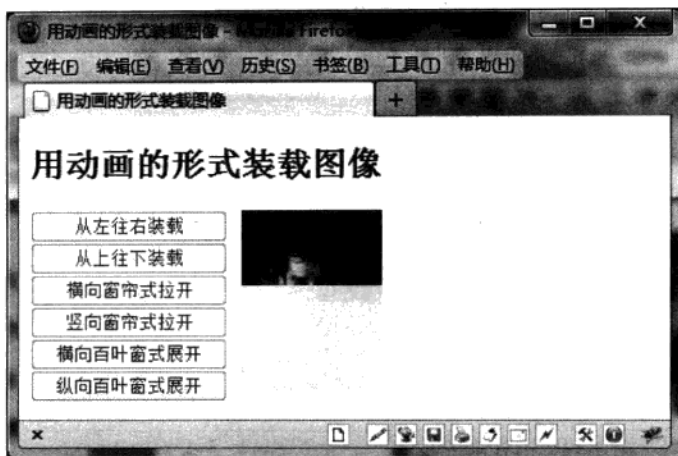


图 3-15 从上往下装载图像

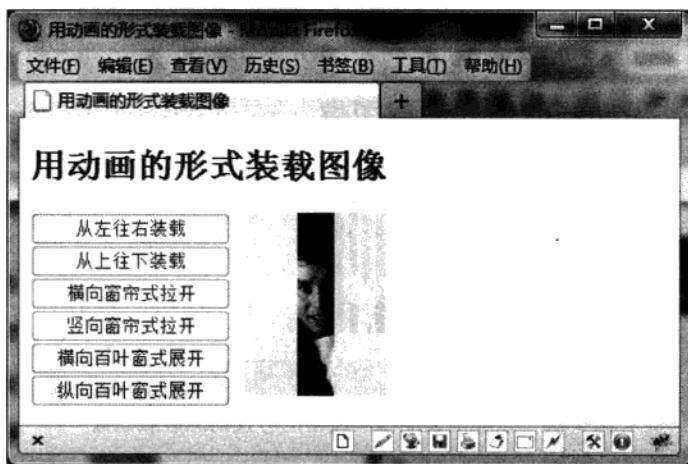


图 3-16 以从中央向左右两边拉开的方式动态装载图像

单击“竖向窗帘式拉开”按钮后，按钮右侧的 canvas 元素开始装载图像，装载方式为首先显示图像中央部分，然后以向上下两边逐渐拉开的方式动态装载图像，如图 3-17 所示，直到图像完全装载完毕。在装载的同时，把所有按钮设为无效状态。图像全部装载完毕后，所有按钮恢复为有效状态。

单击“横向百叶窗式展开”按钮后，按钮右侧的 canvas 元素开始装载图像，装载方式为将图像横向分为很多栅格，以逐渐装满这些栅格的形式装载图像，如图 3-18 所示，直到图像完全装载完毕。在装载的同时，把所有按钮设为无效状态。图像全部装载完毕后，所有按钮



恢复为有效状态。



图 3-17 以从中央向上下两边拉开的方式动态装载图像



图 3-18 以逐渐装满横向栅格的方式动态装载图像

单击“纵向百叶窗式展开”按钮后，按钮右侧的 canvas 元素开始装载图像，装载方式为将图像纵向分为很多栅格，以逐渐装满这些栅格的形式装载图像，如图 3-19 所示，直到图像完全装载完毕。在装载的同时，把所有按钮设为无效状态。图像全部装载完毕后，所有按钮恢复为有效状态。

### 3.5.3 案例知识点

在本案例中，因为使用动态方式来装载图像，每次只装载图像中的一部分，直到装载完毕，所以大量使用了 drawImage 方法中的第三种方法来绘制图像，代码如下所示。



图 3-19 以逐渐装满纵向栅格的方式动态装载图像

```

/* 从左往右装载图像, drawWidth 表示每次绘制的图像宽度,
逐渐增加该值 (从 0 到图像宽度) 来实现绘制效果 */
context.drawImage(image, 0, 0, drawWidth, image.height, 0, 0, drawWidth,
image.height);

/* 从上往下装载图像, drawHeight 表示每次绘制的图像高度,
逐渐增加该值 (从 0 到图像高度) 来实现绘制效果 */
context.drawImage(image, 0, 0, image.width, drawHeight, 0, 0, image.width,
drawHeight);

/* 以从中央向左右两边拉开的方式动态装载图像, drawLeft 表示图像在坐标轴 X 轴上的绘制起点
drawWidth 表示每次绘制的图像宽度, 采用同时减少 drawLeft 值与增加 drawWidth 值的方式来
实现绘制效果, drawLeft 值从 canvas 元素的宽度 / 2 (中心点) 减少到 0, drawWidth 值从 0
增加到 canvas 元素的宽度 */
context.drawImage(image, drawLeft, 0, drawWidth, image.height, drawLeft, 0,
drawWidth, image.height);

/* 以从中央向上下两边拉开的方式动态装载图像, drawTop 表示图像在坐标轴 Y 轴上的绘制起点
drawHeight 表示每次绘制的图像高度, 采用同时减少 drawTop 值与增加 drawHeight 值的方式
来实现绘制效果, drawTop 值从 canvas 元素的高度 / 2 (中心点) 减少到 0, drawHeight 值
从 0 增加到 canvas 元素的高度 */
context.drawImage(image, 0, drawTop, image.width, drawHeight, 0, drawTop,
image.width, drawHeight);

/* 以逐渐装满横向栅格的方式动态装载图像, i 值为 0 到 9, 表示将 canvas 横向分为 10 格,
spaceWidth 值为每格宽度 (canvas 元素宽度 / 10), drawWidth 值表示每次绘制的图像宽度,
通过将该值从 0 增加到 spaceWidth 值的方式逐渐填满栅格 */
context.drawImage(image, 0 + i * spaceWidth, 0, drawWidth, image.height,
0 + i * spaceWidth, 0, drawWidth, image.height);

/* 以逐渐装满纵向栅格的方式动态装载图像, i 值为 0 到 9, 表示将 canvas 纵向分为 10
格, spaceHeight 值为每格高度 (canvas 元素高度 / 10), drawHeight 值表示每次绘制的图像高度,

```

```

    通过将该值从 0 增加到 spaceHeight 值的方式逐渐填满栅格 */
    context.drawImage(image,0,0+i*spaceHeight,image.width,drawHeight,
    0,0+i*spaceHeight,image.width,drawHeight);

```

### 3.5.4 代码剖析

最后来看一下该案例程序的完整代码，如代码清单 3-8 所示，稍后将对这个案例的代码进行详细分析。

代码清单 3-8 用动画的形式装载图像

---

```

<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title> 用动画的形式装载图像 </title>
<script type="text/javascript">
var width,height;
var context;
var image;
var functionId;
var drawLeft;
var drawWidth;
var drawTop;
var drawHeight;
var spaceWidth,spaceHeight;
function window_onload()
{
    var canvas = document.getElementById('canvas');
    context = canvas.getContext('2d');
    image = new Image();
    image.src = "tyl.jpg";
    width=canvas.width;
    height=canvas.height;
}

// 从左往右装载图像
function btn1_onclick()
{
    context.fillStyle = "#EEEEFF";
    context.fillRect(0, 0,width,height);

    drawWidth=0;
    functionId=self.setInterval("drawImg1()",100);
    btnDisable();
}

function drawImg1()
{
    context.drawImage(image,0,0,drawWidth,image.height,0,0,drawWidth,
    image.height);

```

```

drawWidth=drawWidth+2;
if (drawWidth>width)
{
    window.clearInterval(functionId);
    btnEnable();
}
}
// 从上往下装载图像
function btn2_onclick()
{
    context.fillStyle = "#EEEEFF";
    context.fillRect(0, 0,width,height);
    drawHeight=0;
    functionId=self.setInterval("drawImg2()",100);
    btnDisable();
}
function drawImg2()
{
    context.drawImage(image,0,0,image.width,drawHeight,0,0,image.width,
drawHeight);
    drawHeight=drawHeight+2;
    if (drawHeight>height)
    {
        window.clearInterval(functionId);
        btnEnable();
    }
}
// 以从中央向左右两边拉开的方式动态装载图像
function btn3_onclick()
{
    context.fillStyle = "#EEEEFF";
    context.fillRect(0, 0,width,height);
    drawLeft=width/2;
    drawWidth=0;
    functionId=self.setInterval("drawImg3()",100);
    btnDisable();
}
function drawImg3()
{
    context.drawImage(image,drawLeft,0,drawWidth,image.height,drawLeft,0,
drawWidth,image.height);
    drawLeft=drawLeft-1;
    drawWidth=drawWidth+2;
    if (drawLeft<=0)
    {
        window.clearInterval(functionId);
        btnEnable();
    }
}
// 以从中央向上下两边拉开的方式动态装载图像

```

```

function btn4_onclick()
{
    context.fillStyle = "#EEEEFF";
    context.fillRect(0, 0,width,height);
    drawTop=height/2;
    drawHeight=0;
    functionId=self.setInterval("drawImg4()",100);
    btnDisable();
}
function drawImg4()
{
    context.drawImage(image,0,drawTop,image.width,drawHeight,0,drawTop,
        image.width,drawHeight);
    drawTop=drawTop-1;
    drawHeight=drawHeight+2;
    if(drawTop<=0)
    {
        window.clearInterval(functionId);
        btnEnable();
    }
}
// 以逐渐装满横向栅格的方式动态装载图像
function btn5_onclick()
{
    context.fillStyle = "#EEEEFF";
    context.fillRect(0, 0,width,height);
    spaceWidth=width/10;
    drawWidth=0;
    functionId=self.setInterval("drawImg5()",100);
    btnDisable();
}
function drawImg5()
{
    for(i=0;i<10;i++)
    {
        context.drawImage(image,0+i*spaceWidth,0,drawWidth,image.height,
            0+i*spaceWidth,0,drawWidth,image.height);
    }

    drawWidth+=1;

    if(drawWidth>spaceWidth)
    {
        window.clearInterval(functionId);
        btnEnable();
    }
}
// 以逐渐装满纵向栅格的方式动态装载图像
function btn6_onclick()
{

```

```

context.fillStyle = "#EEEEFF";
context.fillRect(0, 0,width,height);
spaceHeight=height/10;
drawHeight=0;
functionId=self.setInterval("drawImg6()",100);
btnDisable();
}
function drawImg6()
{
    for(i=0;i<10;i++)
    {
        context.drawImage(image,0,0+i*spaceHeight,image.width,drawHeight,
            0,0+i*spaceHeight,image.width,drawHeight);
    }

    drawHeight+=1;

    if(drawHeight>spaceHeight)
    {
        window.clearInterval(functionId);
        btnEnable();
    }
}
// 使所有按钮变为无效状态
function btnDisable()
{
    document.getElementById("btn1").disabled="disabled";
    document.getElementById("btn2").disabled="disabled";
    document.getElementById("btn3").disabled="disabled";
    document.getElementById("btn4").disabled="disabled";
    document.getElementById("btn5").disabled="disabled";
    document.getElementById("btn6").disabled="disabled";
}
// 使所有按钮变为有效状态
function btnEnable()
{
    document.getElementById("btn1").disabled="";
    document.getElementById("btn2").disabled="";
    document.getElementById("btn3").disabled="";
    document.getElementById("btn4").disabled="";
    document.getElementById("btn5").disabled="";
    document.getElementById("btn6").disabled="";
}
</script>
<style>
article{
    align:center;
}
canvas{
    background-color:white;

```

```

    }
    div#divLeft{
        width:150px;
        float:left;
    }
    div#divRight{
        float:left;
    }
    input[type='button']{
        width:140px;
    }
</style>
</head>
<body onload="window_onload()">
<article>
<h1> 用动画的形式装载图像 </h1>
<div id="divLeft">
<input type="button" id="btn1" value=" 从左往右装载 "
onclick="btn1_onclick()" />
<input type="button" id="btn2" value=" 从上往下装载 "
onclick="btn2_onclick()" />
<input type="button" id="btn3" value=" 横向窗帘式拉开 "
onclick="btn3_onclick()" />
<input type="button" id="btn4" value=" 竖向窗帘式拉开 "
onclick="btn4_onclick()" />
<input type="button" id="btn5" value=" 横向百叶窗式展开 "
onclick="btn5_onclick()" />
<input type="button" id="btn6" value=" 纵向百叶窗式展开 "
onclick="btn6_onclick()" />
</div>
<div id="divRight">
<canvas id="canvas" width="100" height="130"></canvas>
</div>
</article>
</body>
</html>

```

本案例的 HTML 页面代码部分比较简单，使用一个 h1 元素显示页面标题文字“用动画的形式装载图像”。标题元素的下部分为左右两个区域，左边区域中使用类型为 Button 的 input 元素来显示几个按钮，按钮文字分别为“从左往右装载”、“从上往下装载”、“横向窗帘式拉开”、“竖向窗帘式拉开”、“横向百叶窗式展开”与“纵向百叶窗式展开”。右边的区域中放置一个 canvas 元素。单击左边的按钮后在该 canvas 元素中显示一幅图像，根据单击按钮的不同而以不同的方式来动态装载图像。

接下来针对案例程序的 JavaScript 脚本代码进行详细分析。

脚本代码的开头几行中定义了几个本案例中所使用的全局变量，这些全局变量的含义如下。

- width 与 height: 案例页面中 canvas 元素的宽度与高度。
  - context: 页面中 canvas 元素的图形上下文对象。
  - image: 用来装载图像的 JavaScript 中的 Image 对象。
  - functionId: 用来停止图像装载动画的整型变量。
  - drawLeft 与 drawWidth: 在 canvas 元素中绘制图像时的起始坐标点的横坐标值与所绘制图像的宽度。
  - drawTop 与 drawHeight: 在 canvas 元素中绘制图像时的起始坐标点的纵坐标值与所绘制图像的高度。
  - spaceWidth 与 spaceHeight: 在以横向百叶窗式展开图像的动画过程中每一个百叶窗栅格的宽度与以纵向百叶窗式展开图像的动画过程中每一个百叶窗栅格的高度。
- 在定义了全局变量之后, 定义几个本案例程序中所使用的函数。

#### □ window\_onload 函数

打开页面时调用 window\_onload 函数。在该函数中取得页面中 canvas 元素的图形上下文对象并且赋值给全局变量 context, 创建一个 JavaScript 的 Image 对象并且赋值给全局变量 image, 然后指定该 Image 对象的图像源, 取得页面中 canvas 元素的宽度与高度并赋值给全局变量 width 与 height。

#### □ btn1\_onclick 函数与 drawImg1 函数

单击按钮文字为“从左往右装载”的按钮时将调用 btn1\_onclick 函数。该函数首先指定 canvas 元素的背景为浅灰色, 然后指定全局变量 drawWidth 的变量值 (该变量值代表在 canvas 元素中绘制图像时的绘制宽度) 为 0, 接着设定每隔 100 毫秒调用 drawImg1 函数进行 canvas 元素中图像的重绘 (以达到动画效果), 然后调用 btnDisable 函数将页面中所有按钮的有效状态设定为无效状态, 以防止用户在完整图像绘制完毕前再次单击其他按钮。

在 drawImg1 函数中进行 canvas 元素中的绘制工作。该函数的第一行代码绘制图像中从坐标点 (0, 0) 开始到坐标点 (全局变量 drawWidth 的变量值, 所绘图像的完整高度值) 这一个区域中的图像。开始调用 drawImg1 函数时全局变量 drawWidth 的变量值为 0, 所以没有绘制任何图像。在 drawImg1 函数接下来的代码中, 将全局变量 drawWidth 的变量值加上 2, 因此下次绘制的图像宽度比本次绘制的图像宽度多 2 个像素。通过每隔 100 毫秒调用一次 drawImg1 函数, 实现了将指定图像从左往右进行展开的动画效果。当全局变量 drawWidth 的变量值等于 canvas 元素的宽度时, 表示图像展开完毕, 这时调用 window.clearInterval 方法停止图像的展开动画, 同时调用 btnEnable 函数将页面中所有按钮还原为有效状态, 以便用户再次单击其他按钮来查看图像的其他动态装载效果。

#### □ btn2\_onclick 函数与 drawImg2 函数

单击按钮文字为“从上往下装载”的按钮时将调用 btn2\_onclick 函数。该函数首先指定 canvas 元素的背景为浅灰色, 指定 drawHeight 全局变量的变量值 (该变量值代表在 canvas 元素中绘制图像时的绘制高度) 为 0, 然后设定每隔 100 毫秒调用 drawImg2 函数



进行 canvas 元素中图像的重绘（以达到动画效果），接着调用 btnDisable 函数将页面中所有按钮的有效状态设定为无效状态，以防止用户在整个图像绘制完毕前再次单击其他按钮。

在 drawImg2 函数中进行在 canvas 元素中的绘制工作。该函数的第一行代码即为绘制从坐标点 (0, 0) 开始到坐标点（所绘图像的完整宽度值，全局变量 drawHeight 的变量值）这一个区域中的图像。开始调用 drawImg2 函数时，drawHeight 全局变量的变量值为 0，因此没有绘制任何图像。在 drawImg2 函数接下来的代码中，将全局变量 drawHeight 的变量值加 2，因此下次绘制的图像高度比本次绘制的图像高度多 2 个像素。通过每隔 100 毫秒调用一次 drawImg2 函数，实现了将指定图像从上往下进行展开的动画效果。当全局变量 drawHeight 的变量值等于 canvas 元素的高度时，表示图像展开完毕，这时调用 window.clearInterval 方法停止图像的展开，同时调用 btnEnable 函数将页面中所有按钮还原为有效状态，以便用户再次单击其他按钮来查看图像的其他动态装载效果。

#### □ btn3\_onclick 函数与 drawImg3 函数

单击按钮文字为“横向窗帘式拉开”的按钮时将调用 btn3\_onclick 函数。该函数首先指定 canvas 元素的背景为浅灰色，然后指定变量 drawLeft 的变量值为 canvas 元素的宽度值 /2（表示开始在 canvas 元素中绘制图像时绘制点的横坐标为 canvas 元素的中央），指定全局变量 drawWidth 的变量值（该变量值代表在 canvas 元素中绘制图像时的绘制宽度）为 0，然后设定每隔 100 毫秒调用 drawImg3 函数进行 canvas 元素中图像的重绘（以达到动画效果），接下来调用 btnDisable 函数将页面中所有按钮的有效状态设定为无效状态，以防止用户在整个图像绘制完毕前再次单击其他按钮。

在 drawImg3 函数中进行 canvas 元素中的绘制工作。该函数的第一行代码完成绘制从坐标点（全局变量 drawLeft 的变量值，0）开始到坐标点（全局变量 drawWidth 的变量值，所绘图像的完整高度值）这一个区域中的图像。开始调用 drawImg3 函数时全局变量 drawLeft 的变量值为 canvas 元素的宽度值 /2，全局变量 drawWidth 的变量值为 0，因此没有绘制任何图像，绘制起点在 canvas 元素的中央。drawImg3 函数接下来的代码将全局变量 drawLeft 的变量值减 2，全局变量 drawWidth 的变量值加 2，因此下次绘制图像时的绘制起点为本次绘制图像时的绘制起点左移 2 个像素，绘制宽度比本次绘制图像的绘制宽度多 2 个像素。通过每隔 100 毫秒调用一次 drawImg3 函数，实现了将指定图像从中央向两边进行横向展开的动画效果。当全局变量 drawLeft 的变量值小于 0 时，表示图像展开完毕，这时调用 window.clearInterval 方法停止图像的展开，同时调用 btnEnable 函数将页面中所有按钮还原为有效状态，以便用户再次单击其他按钮来查看图像的其他动态装载效果。

#### □ btn4\_onclick 函数与 drawImg4 函数

单击按钮文字为“竖向窗帘式拉开”的按钮时将调用 btn4\_onclick 函数。在该函数中首先指定 canvas 元素的背景为浅灰色，然后指定变量 drawTop 的变量值为 canvas 元素的高度值 /2（表示在 canvas 元素中开始绘制图像时绘制点的纵坐标为 canvas 元素的中央），指定全局变量 drawHeight 的变量值（该变量值代表在 canvas 元素中绘制图像时的绘制高度）为 0，

设定每隔 100 毫秒调用 drawImg4 函数进行 canvas 元素中图像的重绘（以达到动画效果），接着调用 btnDisable 函数将页面中所有按钮设定为无效状态，以防止用户在整个图像绘制完毕前再次单击其他按钮。

在 drawImg4 函数中进行 canvas 元素中的绘制工作。该函数的第一行代码即为绘制从坐标点 (0, 全局变量 drawTop 的变量值) 开始到坐标点 (所绘图像的完整宽度值, 全局变量 drawHeight 的变量值) 这一个区域中的图像。开始调用 drawImg4 函数时全局变量 drawTop 的变量值为 canvas 元素的高度值 /2, 全局变量 drawHeight 的变量值为 0, 因此没有绘制任何图像, 绘制起点在 canvas 元素的中央。drawImg4 函数接下来的代码将全局变量 drawTop 的变量值减 2, 全局变量 drawHeight 的变量值加上 2, 因此下次绘制图像时的绘制起点为本次绘制图像时的绘制起点上移 2 个像素, 绘制高度比本次绘制图像的绘制高度多 2 个像素。通过每隔 100 毫秒调用一次 drawImg4 函数, 实现了将指定图像从中央向两边进行纵向展开的动画效果。当全局变量 drawTop 的变量值小于 0 时, 表示图像展开完毕, 这时调用 window.clearInterval 方法停止图像的展开动画, 同时调用 btnEnable 函数将页面中所有按钮还原为有效状态, 以便用户再次单击其他按钮来查看图像的其他动态装载效果。

#### □ btn5\_onclick 函数与 drawImg5 函数

单击按钮文字为“横向百叶窗式展开”的按钮时将调用 btn5\_onclick 函数。该函数首先指定 canvas 元素的背景为浅灰色, 然后指定变量 spaceWidth 的变量值为 canvas 元素的宽度值 /10 (表示将 canvas 元素横向分为 10 个栅格), 指定全局变量 drawWidth 的变量值 (该变量值代表在 canvas 元素中绘制图像时的绘制宽度) 为 0, 设定每隔 100 毫秒调用 drawImg5 函数进行 canvas 元素中图像的重绘 (以达到动画效果), 然后调用 btnDisable 函数将页面中所有按钮的有效状态设定为无效状态, 以防止用户在整个图像绘制完毕前再次单击其他按钮。

在 drawImg5 函数中进行 canvas 元素中的绘制工作。该函数的前三行代码表示将 canvas 元素横向分为 10 个栅格, 在每一格中绘制从坐标点 (每一个栅格的起始点横坐标值, 0) 到坐标点 (全局变量 drawWidth 的变量值, 所绘图像的完整高度值) 这一个区域中的图像。因为开始调用 drawImg5 函数时全局变量 drawWidth 的变量值为 0, 所以没有绘制任何图像。drawImg5 函数接下来的代码将全局变量 drawWidth 的变量值加 1, 因此下次绘制图像时的绘制宽度比本次绘制图像的绘制宽度多 1 个像素。通过每隔 100 毫秒调用一次 drawImg5 函数, 实现了将指定图像呈横向百叶窗式展开显示的动画效果。当全局变量 drawWidth 的变量值大于代表每一个栅格宽度的变量 spaceWidth 的变量值时, 表示图像展开完毕, 这时调用 window.clearInterval 方法停止图像的展开动画, 同时调用 btnEnable 函数将页面中所有按钮还原为有效状态, 以便用户再次单击其他按钮来查看图像的其他动态装载效果。

#### □ btn6\_onclick 函数与 drawImg6 函数

单击按钮文字为“纵向百叶窗式展开”的按钮时将调用 btn6\_onclick 函数。该函数首先

指定 canvas 元素的背景为浅灰色, 然后指定变量 spaceHeight 的变量值为 canvas 元素的高度值 /10 (表示将 canvas 元素纵向分为 10 个栅格), 指定全局变量 drawHeight 的变量值 (该变量值代表在 canvas 元素中绘制图像时的绘制高度) 为 0, 设定每隔 100 毫秒调用 drawImg6 函数进行 canvas 元素中图像的重绘 (以达到动画效果), 然后调用 btnDisable 函数将页面中所有按钮的有效状态设定为无效状态, 以防止用户在整个图像绘制完毕前再次单击其他按钮。

在 drawImg6 函数中进行 canvas 元素中的绘制工作。该函数的前三行代码表示将 canvas 元素纵向分为 10 个栅格, 在每一格中绘制从坐标点 (0, 每一个栅格的起始点纵坐标值) 到坐标点 (所绘图像的完整宽度值, 全局变量 drawHeight 的变量值) 这一个区域中的图像。因为开始调用 drawImg6 函数时全局变量 drawHeight 的变量值为 0, 所以没有绘制任何图像。drawImg6 函数接下来的代码将全局变量 drawHeight 的变量值加 1, 因此下次绘制图像时的绘制高度比本次绘制图像的绘制高度多 1 个像素。通过每隔 100 毫秒调用一次 drawImg6 函数, 实现了将指定图像呈纵向百叶窗式展开显示的动画效果。当 drawHeight 全局变量的变量值大于代表每一个栅格宽度的变量 spaceHeight 的变量值时, 表示图像展开完毕, 这时调用 window.clearInterval 方法停止图像的展开动画, 同时调用 btnEnable 函数将页面中所有按钮还原为有效状态, 以使用户再次单击其他按钮来查看图像的其他动态装载效果。

## 3.6 案例 10: 将彩色照片转换成黑白照片

### 3.6.1 案例概述

本节通过一个将彩色照片转换成黑白照片的案例来介绍如何使用 canvas 元素对图像的像素进行处理, 然后将转换后的图片输出并保存到用户计算机中。在浏览器中打开案例页面时, 页面中显示一幅彩色照片, 单击页面中的转换按钮, 页面中显示转换后的黑白照片, 单击页面中的保存按钮, 浏览器中的案例页面被关闭, 打开一个新的页面, 页面中只显示转换后的黑白照片, 用户可以将这幅黑白照片保存在计算机中。

### 3.6.2 页面显示效果

本案例页面在浏览器中打开时的显示效果如图 3-20 所示 (使用 Opera 11 浏览器)。案例页面打开时页面中的保存按钮为无效状态。

单击转换按钮后页面下部出现由彩色照片转换成的黑白照片, 同时保存图片按钮变为有效状态。如图 3-21 所示。

单击保存图片按钮后案例页面被关闭, 在浏览器中打开新页面, 页面中只显示转换后的黑白照片, 如图 3-22 所示。用户可以将该图片保存到计算机中。



图 3-20 在 Opera 11 浏览器中打开案例页面时的显示效果



图 3-21 单击转换按钮后页面中出现转换后的黑白照片

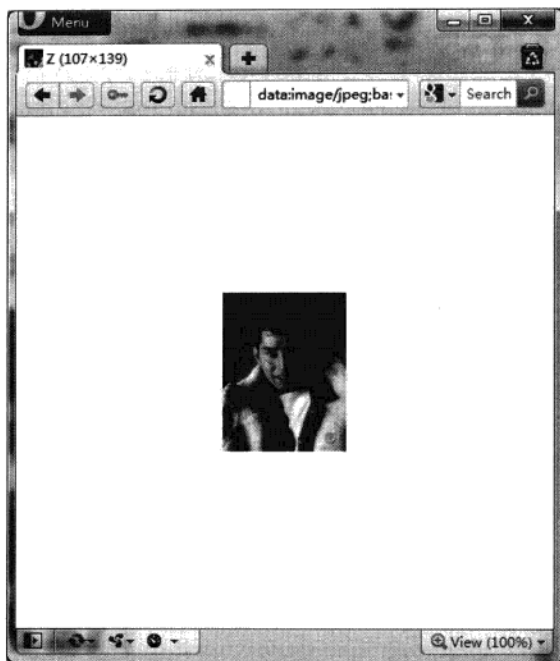


图 3-22 单击保存图片按钮后浏览器中出现转换后的黑白照片

### 3.6.3 案例知识点

#### 1. 对图像的像素进行处理

在 HTML 5 的 Canvas API 图像处理技术中，还有一个令人赞叹的技术是像素处理技术。使用 Canvas API 能够获取图像中的每一个像素，然后得到该像素颜色的 rgb 值或 rgba 值。使用图形上下文对象的 `getImageData` 方法来获取图像中的像素，该方法的定义如下。

```
var imagedata= context.getImageData(sx, sy, sw, sh);
```

该方法使用 4 个参数，`sx` 和 `sy` 分别表示所获取区域的起点横坐标和起点纵坐标，`sw`, `sh` 分别表示所获取区域的宽度和高度。

`imagedata` 变量是一个 `CanvasPixelArray` 对象，具有 `height`、`width` 和 `data` 等几个属性。`data` 属性是一个保存像素数据的数组，内容类似 “[ `r1`, `g1`, `b1`, `a1`, `r2`, `g2`, `b2`, `a2`, `r3`, `g3`, `b3`, `a3`, ... ]”，`r1`、`g1`、`b1` 和 `a1` 为第一个像素的红色值、绿色值、蓝色值和透明值，`r2`、`g2`、`b2` 和 `a2` 分别为第二个像素的红色值、绿色值、蓝色值和透明值，依此类推。`data.length` 为所取得像素的数量。

在 Canvas API 中，使用图形上下文对象的 `putImageData` 方法将经过像素处理操作后的图像重新绘制在 `canvas` 元素中。该方法的定义如下。

```
context.putImageData(imagedata,dx, dy [, dirtyX, dirtyY, dirtyWidth,
dirtyHeight ]);
```

该方法有 7 个参数, imagedata 为前面所述的像素数组, dx 和 dy 分别表示重绘图像的起点横坐标和起点纵坐标。后面 dirtyX、dirtyY、dirtyWidth 和 dirtyHeight 这 4 个参数为可选参数, 给出一个矩形的起点横坐标、起点纵坐标、宽度和高度, 如果加上这 4 个参数, 则只绘制像素数组中这个矩形范围内的图像。

另外, Canvas API 的像素操作也只被部分浏览器支持, 所以笔者选择 Opera 11 浏览器进行代码测试。

## 2. 保存 canvas 元素中的图像

在 canvas 元素中绘制完成一幅图形或图像后, 很多时候需要将该图形或图像保存到计算机中, 使用 Canvas API 当然可以完成这最后一步工作。

Canvas API 保存文件的原理实际上是一步把当前的绘画状态输出到一个 data URL 地址所指向的数据中, 所谓 data URL, 是指目前大多数浏览器能够识别的一种基于 64 位编码的 URL, 主要用于小型的, 可以在网页中直接嵌入, 而不需要从外部文件嵌入的数据, 譬如 img 元素中的图像文件等。dataURL 的格式类似于 “data:image/png;base64,iVBORw0KGgoA AAANSUUhEUgAAAAoAAAAK...etc”, 目前受到了大多数浏览器的支持。

Canvas API 使用 toDataURL 方法把绘画状态输出到一个 dataURL, 然后重新装载, 客户可直接将装载后的文件保存。

toDataURL 的使用方法如下所示。

```
canvas.toDataURL (type);
```

该方法有一个参数 type, 表示要输出数据的 MIME 类型。

## 3.6.4 代码剖析

最后来看一下该案例程序的完整代码, 如代码清单 3-9 所示, 稍后将对这个案例的代码进行详细分析。

代码清单 3-9 将彩色照片转换成为黑白照片的完整代码

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript">
var canvas,ctx;
function btnConvert_onclick() {
    canvas = document.getElementById("myCanvas");
    var imgElement=document.getElementById("img");
    canvas.width=imgElement.width;
    canvas.height=imgElement.height;
```

```

    ctx = canvas.getContext("2d");
    imgElement.onload = function() {
        ctx.drawImage(imgElement, 0, 0);
        imageConvertToGray();
    }
    imgElement.src = "tyl.jpg";
    document.getElementById("btnSave").disabled="";
}
function imageConvertToGray() {
    var length = canvas.width * canvas.height;
    imageData = ctx.getImageData(0, 0, canvas.width, canvas.height);
    for (var i = 0; i < length * 4; i += 4) {
        var myRed = imageData.data[i];
        var myGreen = imageData.data[i + 1];
        var myBlue = imageData.data[i + 2];
        myGray = parseInt((myRed + myGreen + myBlue) / 3);
        imageData.data[i] = myGray;
        imageData.data[i + 1] = myGray;
        imageData.data[i + 2] = myGray;
    }
    ctx.putImageData(imageData, 0, 0);
}
function btnSave_onclick()
{
    window.location =canvas.toDataURL("image/jpeg");
}
</script>
</head>
<body>
<h1> 将彩色照片转换成黑白照片 </h1>
<br/>
<input type="button" id="btnConvert" value=" 转换 " onclick="btnConvert_onclick();"/>
<input type="button" id="btnSave" value=" 保存图片 " onclick="btnSave_onclick();"
disabled/><br/>
<canvas id="myCanvas" width="200" height="200"/>
</body>
</html>

```

接下来针对案例程序的 JavaScript 脚本代码进行详细分析。

脚本代码的第一行定义了本案例中所使用的两个全局变量，其中变量 canvas 表示页面中使用的 canvas 元素，变量 ctx 表示该 canvas 元素的图形上下文对象。

在定义了全局变量之后，定义本案例程序中所使用的几个函数。

#### ❑ btnConvert\_onclick 函数

在用户单击转换按钮时调用 btnConvert\_onclick 函数将彩色照片转换为黑白照片。在该函数中首先获取页面上的 canvas 元素并将其赋值给 canvas 变量，获取页面上用来显示彩

色照片的 `img` 元素并将其赋值给变量 `imgElement`。同时将 `canvas` 元素的宽度与高度设置为 `img` 元素的宽度与高度（即彩色照片的宽度与高度，使黑白照片的宽度与高度和彩色照片的宽度与高度相等）。接下来获取 `canvas` 元素的图形上下文对象并将其赋值给变量 `ctx`，然后重新指定页面中 `img` 元素中显示的彩色照片来源并重新在 `img` 元素中装载彩色照片，装载的同时将彩色照片绘制在 `canvas` 元素中，并使用 `imageConvertToGray` 函数将 `canvas` 元素中彩色照片的像素转换为黑白照片的像素，之后重新绘制在 `canvas` 元素中，在函数的结尾处将页面中的保存按钮设定为有效状态。

#### ❑ `imageConvertToGray` 函数

在 `imageConvertToGray` 函数中，将 `canvas` 元素中的彩色照片的像素转换为黑白照片的像素，然后重新绘制在 `canvas` 元素中。在该函数的开头，将 `canvas` 元素的宽与高的乘积赋值给变量 `length`，也就是使变量 `length` 的值等于照片中所有像素的个数。然后使用 `getImageData` 方法来获取图像中所有像素并赋值给 `imageData` 对象。接下来，使用变量 `i` 来循环遍历 `imageData` 对象的 `data` 属性，该属性值为一个数组，数组中各选项内容为：图像中第一个像素的红色值（即 `rgba` 颜色中的 `r` 值）、图像中第一个像素的绿色值（即 `rgba` 颜色中的 `g` 值）、图像中第一个像素的蓝色值（即 `rgba` 颜色中的 `b` 值）、图像中第一个像素的透明度（即 `rgba` 颜色中的 `a` 值）；图像中第二个像素的红色值、图像中第二个像素的绿色值、图像中第二个像素的蓝色值、图像中第二个像素的透明度；……；图像中最后一个像素的红色值、图像中最后一个像素的绿色值、图像中最后一个像素的蓝色值、图像中最后一个像素的透明度。循环的终点值为变量 `length` 的值 `*4`，这是因为 `imageData` 对象的 `data` 属性值数组的大小为像素的个数 `*4`（每个像素占据数组中的 4 个数组项）；循环的下一步中 `i` 值为当前步中 `i` 值 `+4`，这是因为在每一步中对每个像素所占据的 4 个数组项进行统一处理。在循环中首先取出每个像素的红色值赋值给变量 `myRed`，取出每个像素的绿色值赋值给变量 `myGreen`，取出每个像素的蓝色值赋值给变量 `myBlue`，计算每个变量的红色值、绿色值与蓝色值的平均值并赋值给变量 `myGray`，然后将每个像素的红色值、绿色值与蓝色值都设定为变量 `myGray` 的值，循环结束后使用 `canvas` 元素的图形上下文对象的 `putImageData` 方法将经过像素处理操作后的图像重新绘制在 `canvas` 元素中。

#### ❑ `btnSave_onclick` 函数

用户单击保存按钮时调用 `btnSave_onclick` 函数。在该函数中使用 `toDataURL` 方法把 `canvas` 元素中的黑白图像重新输出到新的网页中并将该网页打开，用户可以在该网页中将图像保存到计算机中。

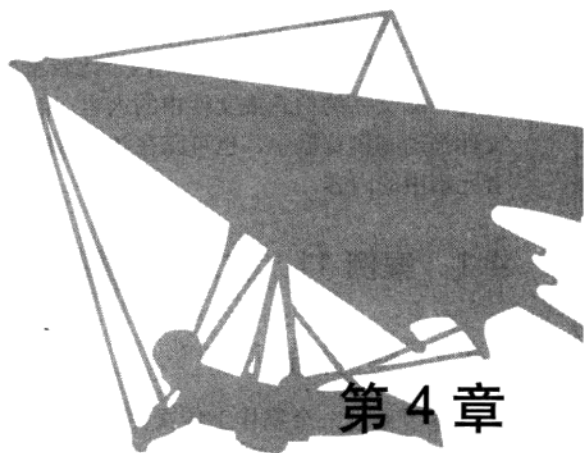
## 3.7 本章小结

本章通过 6 个案例，详细阐述了如何利用 HTML 5 中的 `canvas` 元素与 Canvas API 在网页上绘制图形、图像与动画，以及制作小游戏，力求使读者通过阅读本章，对于 HTML 5 中的 `canvas` 元素与 Canvas API 的基本知识与使用方法有一个比较细致的了解，能够利用 `canvas`



元素与 Canvas API 在页面上绘制出属于自己的图形、图像以及动画。

下一章将通过一些案例来具体讲解如何利用 HTML 5 中的文件 API 来读取客户端计算机中的文件内容和文件信息，如何利用 HTML 5 中的拖放 API 在 Web 页面中正确读取用户从计算机中拖动到页面中的内容，以及如何通过拖放 API 来允许用户将一个元素中的内容或子元素拖动到页面的另一个元素中。



## 第 4 章

# 文件 API 与拖放 API

### 本章内容

- 案例 11：在浏览器中预览客户端文件并上传
- 案例 12：使用 Canvas API、文件 API 与拖放 API 制作拼图游戏
- 本章小结

本章通过两个案例向读者展示 HTML 5 中文件 API 与拖放 API 的使用方法。在 HTML 5 中，文件 API 与拖放 API 也是两个比较重要的 API。通过文件 API，可以在浏览器中直接显示客户端文件的信息或文件中的内容，而通过拖放 API，可以直接将位于客户端计算机中的文件拖动到浏览器中，也可以在浏览器的同一网页中，或不同浏览器中拖动页面中的元素或者元素中的内容。

## 4.1 案例 11：在浏览器中预览客户端文件并上传

### 4.1.1 案例概述

本节通过一个制作上传文件的案例向读者展示 HTML 5 中文件 API 与拖放 API 的使用方法。在本案例中，用户上传文件的操作方法与用户在传统网页中执行文件上传操作的方法不同。在目前的 HTML 4 版网页中，用户使用浏览器所显示页面中的 file 控件，单击上传按钮后选择计算机中的文件，然后执行文件上传操作。这样做的不足之处是，必须先确定后文件的内容，然后才能执行文件上传操作。在本案例中，用户可以先将计算机中的文件直接拖动到浏览器中进行预览，待确定某个文件是所需要上传的文件后，单击上传按钮将该文件上传到服务器端。

另外，本案例使用 AJAX 上传文件方式，同时为了实现文件上传功能，使用 Firefox 浏览器提供的 file 对象的 getAsBinary 方法与 XMLHttpRequest 对象的 sendAsBinary 方法，因此本案例仅在 Firefox 浏览器中能够得到完全实现。在 Chrome 浏览器或 Opera 浏览器中，仅能够实现本案例中将文件从客户端计算机中拖放到浏览器中进行预览的功能。

并不是所有文件都能够被预览，因此本案例只提供对于图片文件与文本文件的预览功能，也就是说，本案例只能提供图片文件上传或文本文件上传的功能。

### 4.1.2 页面显示效果

接下来先来看一下该页面在浏览器中的显示效果。

该页面中有一个“文件预览并上传”的标题文字、一个文件预览区域以及一个上传按钮，页面打开时上传按钮为无效状态，如图 4-1 所示。

用户可以将磁盘文件拖动到文件预览区中，如图 4-2 所示。

如果用户拖动的是图像文件，则拖动完毕后文件预览区中显示此图像，同时上传按钮变为有效状态，如图 4-3 所示。

如果用户拖动的是文本文件，则拖动完毕后文件预览区



图 4-1 页面打开时的显示效果

中显示此文本文件的内容，同时上传按钮变为有效状态，如图4-4所示。

如果用户拖动的是其他文件，则拖动完毕后文件预览区中显示文字“暂不支持此类文件的预览”，提示用户不支持其他文件的预览功能，此时上传按钮仍然为无效状态，如图4-5所示。



图 4-2 用户拖动磁盘文件到文件预览区中



图 4-3 拖动图像文件完毕后文件预览区中显示此图像



图 4-4 拖动文本文件完毕后文件预览区中显示文本内容



图 4-5 拖动其他文件完毕后文件预览区中显示文字“暂不支持此类文件的预览”

另外，用户可能将比较大的图像文件或文本文件拖放到文件预览区中，从而导致图像文件的尺寸或文本文件的显示范围超出文件预览区的尺寸。针对这种情况，为文件预览区所使用的 div 元素指定 `resize` 样式，这样用户可以通过拖动文件预览区的右下角来放大文件预览区，使其能够完全显示其中的图片或内容，如图 4-6 所示。

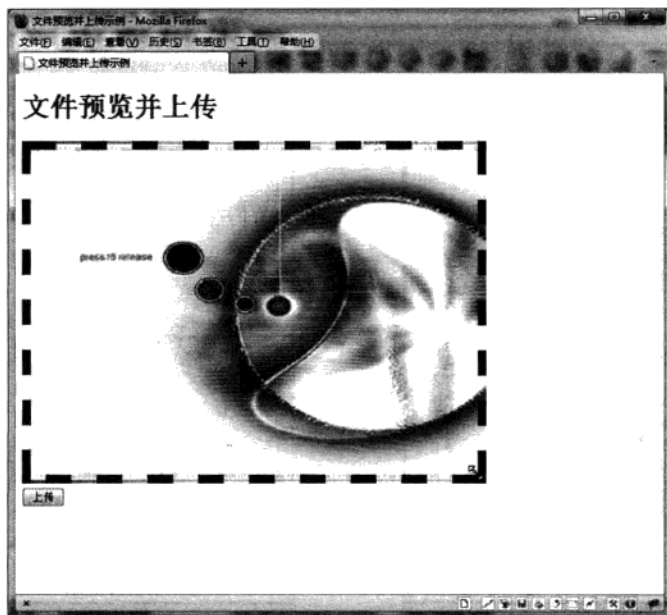


图 4-6 用户自行拖曳文件预览区的尺寸

用户拖动文件完毕后，单击上传按钮，上传成功后页面中显示文字“上传成功”，文件预览区恢复显示文字“文件预览区”，同时上传按钮文字变为“继续上传”，按钮状态恢复为无效状态，如图 4-7 所示。

### 4.1.3 案例知识点

本案例同时使用到了 HTML 5 中的两个 API——文件 API 与拖放 API。接下来对本案例中所使用到的这两个 API 的有关知识进行介绍，关于这两个 API 的详细介绍，请阅读笔者所著的《HTML 5 与 CSS 3 权威指南》一书。

#### 1. 文件 API

HTML5 提供了一个关于文件操作的文件 API，通过这个 API，使从 Web 页面上访问本地文件系统的相关处理变得十分简单。

本案例主要使用了文件 API 中的 file 对象与 FileReader 接口。

在 HTML 5 中，file 对象代表客户端计算机中的一个文件，该对象具有两个属性，name 属性表示文件名（不包括路径）lastModifiedDate 属性表示文件的最后修改日期。

在 HTML 5 中，可以使用 FileReader 接口把文件读入内存，并且读取文件中数据。

FileReader 接口拥有 4 个方法，其中 3 个用于读取文件，另一个用于将读取过程中断。表 4-1 列出了这 4 个方法以及它们的参数和功能。需要注意的是：无论读取成功或失败，方法并不会返回读取结果，这一结果存储在 result 属性中。

表 4-1 FileReader 接口的方法

方法名	参数	描述
readAsBinaryString	file	将文件读取为二进制码
readAsText	file, [encoding]	将文件读取为文本
readAsDataURL	file	将文件读取为 DataURL
abort	(none)	中断读取操作

下面具体介绍 FileReader 接口的 4 个方法。

- ❑ readAsBinaryString：这个方法将文件读取为二进制字符串，将该字符串传送到后端，后端可以通过这段字符串存储文件。
- ❑ readAsText：此方法有两个参数，其中第二个参数是文本的编码方式，默认值为 UTF-8。这个方法非常容易理解，将文件以文本方式读取，读取的结果即是这个文本



图 4-7 文件上传成功后显示“上传成功”文字

文件中的内容。

- ❑ `readAsDataURL`：此方法将文件读取为一串 Data URL 字符串。事实上此方法将小文件以一种特殊格式的 URL 地址形式直接读入页面。这里的小文件通常是指图像与 HTML 等格式的文件。

除了以上方法之外，`FileReader` 接口还包含一套完整的事件模型，其中的事件用于捕获读取文件时的状态，表 4-2 归纳了这些事件。

表 4-2 `FileReader` 接口的事件

事件	描述
<code>onabort</code>	数据读取中断时触发
<code>onerror</code>	数据读取出错时触发
<code>onloadstart</code>	数据读取开始时触发
<code>onprogress</code>	数据读取中
<code>onload</code>	数据读取成功完成时触发
<code>onloadend</code>	数据读取完成时触发，无论成功或失败

## 2. 拖放 API

HTML 5 提供了直接支持拖放操作的 API。虽然在 HTML 5 之前已经可以使用 `mousedown`、`mousemove` 和 `mouseup` 来实现拖放操作，但是这些事件只支持在浏览器内部的拖放，而 HTML 5 已经可以支持浏览器与其他应用程序之间的数据的互相拖动，同时大大简化了拖放方面的代码。

要想在 HTML5 中实现拖放操作，至少要经过如下两个步骤：

- 1) 将要拖放的对象元素的 `draggable` 属性设为 `true`(`draggable="true"`)，这样才能对该元素进行拖放。另外，默认允许拖放 `img` 元素与 `a` 元素（必须指定 `href`）。
- 2) 编写与拖放有关的事件处理代码。拖放的几个事件如表 4-3 中所示。

表 4-3 拖放的相关事件

事件	产生事件的元素	描述
<code>dragstart</code>	被拖放的元素	开始拖放操作
<code>drag</code>	被拖放的元素	拖放过程中
<code>dragenter</code>	拖放过程中鼠标经过的元素	被拖放的元素开始进入本元素的范围内
<code>dragover</code>	拖放过程中鼠标经过的元素	被拖放的元素正在本元素范围内移动
<code>dragleave</code>	拖放过程中鼠标经过的元素	被拖放的元素离开本元素的范围
<code>drop</code>	拖放的目标元素	有其他元素被拖放到了本元素中
<code>dragend</code>	拖放的对象元素	拖放操作结束

另外，本章还利用了拖放 API 中的一个 `DataTransfer` 对象，该对象专门用来保存拖放时所携带的数据。本章使用该对象的 `files[0]` 属性来引用被拖动到文件预览区域中的文件，

代码如下所示。

```
var file = ev.dataTransfer.files[0];
```

#### 4.1.4 代码剖析

接下来具体分析本案例的整个实现过程及实现代码。

##### 1. HTML 页面代码

本案例页面的 HTML 页面代码部分相对比较简单，页面中使用一个表单来实现文件上传功能，当用户单击上传按钮时将该表单进行 AJAX 提交。使用一个 h1 元素显示标题文字“文件预览并上传”，使用一个 div 元素显示文件预览区域，使用一个 submit 按钮作为上传按钮，使用一个 div 元素显示“上传成功”文字，页面代码如代码清单 4-1 所示。

代码清单 4-1 案例的 HTML 页面代码部分

---

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title> 文件预览并上传示例 </title>
<style>
... 样式代码部分稍后介绍 ...
</style>
<script type="text/javascript">
... 脚本代码部分稍后介绍 ...
</script>
</head>
<form id="form1" method="post" action="javascript: uploadAndSubmit();">
<h1> 文件预览并上传 </h1>
<div id="main">
    <div id="show">
        文件预览区
    </div>
</div>
<input type="submit" value=" 上传 " id="saveButton" disabled="disabled"/>
<div id="successLabel" />
</form>
</body>
```

---

##### 2. 样式代码

接下来分析该案例中的样式代码部分。案例的样式代码如代码清单 4-2 中所示。

代码清单 4-2 案例的样式代码部分

---

```
<style>
div[id*=show] {
    border: 10px dashed #ccc;
    width: 300px;
```

---



```
height: 300px;
margin-top: 5px;
margin-bottom: 5px;
resize: both;
overflow: auto;
}
div[id^=show]:hover{
border: 10px dashed #333;
}
div#main{
width: 100%;
}
div#successLabel
{
color: Red;
}
</style>
```

---

在该样式代码中，值得注意的是，将显示文件预览区的 div 元素（id 为 show）的 overflow 属性的属性值设定为 auto，同时将 resize 属性的属性值设定为 both，这样用户就可以通过拖曳文件预览区的右下角来动态改变文件预览区的大小。将文件拖动到文件预览区中，如果默认的文件预览区的尺寸容纳不下被拖动文件的图像或文本文件的内容，此时用户可以通过拖曳文件预览区域来动态改变文件预览区的尺寸，使其能够完整显示被拖动的图像或文本文件中的内容。

### 3. JavaScript 脚本代码

接下来分析本案例中的 JavaScript 脚本代码部分。脚本代码中主要有两个函数，在页面打开时调用 init 函数来设置文件预览区，使其能够接收文件拖放，当图像文件或者文本文件被拖入时显示被拖入的图像或文本文件的内容。当用户单击上传按钮时调用 uploadAndSubmit 函数将文件内容进行 AJAX 上传。

接下来通过代码清单 4-3 来了解本案例中完整的 JavaScript 脚本代码，之后对这个脚本代码进行详细分析。

代码清单 4-3 案例的 JavaScript 脚本代码部分

---

```
<script type="text/javascript">
1   var sendInformation = new Object();
2   function init()
3   {
4       var dest = document.getElementById("show");
5       dest.addEventListener("dragover", function(ev)
6       {
7           ev.stopPropagation();
8           ev.preventDefault();
9       }, false);
```

```

10     dest.addEventListener("dragend", function(ev)
11     {
12         ev.stopPropagation();
13         ev.preventDefault();
14     }, false);
15
16     dest.addEventListener("drop", function (ev) {
17         ev.stopPropagation();
18         ev.preventDefault();
19
20         var file = ev.dataTransfer.files[0];
21         var reader = new FileReader();
22         sendInformation.fileName = file.name;
23         var saveButton = document.getElementById("saveButton");
24
25         if (file.type.substr(0, 5) == "image") {
26             reader.onload = function (event) {
27                 dest.style.background = 'url(' + event.target.result +
28                     ') no-repeat center';
29                 dest.innerHTML = "";
30             };
31             reader.readAsDataURL(file);
32             sendInformation.content = file.getAsBinary();
33
34             sendInformation.fileType = 1;
35             saveButton.disabled = "";
36         }
37         else if (file.type.substr(0, 4) == "text") {
38             reader.readAsText(file);
39             reader.onload = function (f) {
40                 dest.innerHTML = "<pre>" + this.result + "</pre>";
41                 sendInformation.content = this.result;
42                 dest.style.background = "white";
43             }
44             saveButton.disabled = "";
45             sendInformation.fileType = 2;
46         }
47         else {
48             dest.innerHTML = "暂不支持此类文件的预览";
49             dest.style.background = "white";
50             saveButton.disabled = "disabled";
51             sendInformation.fileType = 3;
52         }
53         document.getElementById("successLabel").innerHTML = "";
54     }, false);
55
56     function uploadAndSubmit() {
57         var xhr = new XMLHttpRequest();
58         xhr.open("POST", "dragAndSaveBack.aspx?fileName=" +

```

```

        encodeURIComponent(sendInformation.fileName) + "&fileType=" +
        sendInformation.fileType);

59
60     if(sendInformation.fileType==1)
61         xhr.sendAsBinary(sendInformation.content);
62     else if(sendInformation.fileType==2)
63         xhr.send(sendInformation.content);
64
65     xhr.onreadystatechange =
66     function () {
67         var result = xhr.responseText;
68         document.getElementById("show").innerHTML = " 文件预览区 ";
69         document.getElementById("show").style.background = "white";
70         document.getElementById("successLabel").innerHTML = result;
71         document.getElementById("saveButton").disabled = "disabled";
72
73         if (xhr.readyState == 4) {
74             if (result == " 上传成功 ")
75                 document.getElementById("saveButton").value = " 继续上传 ";
76             else
77                 document.getElementById("saveButton").value = " 重新上传 ";
78         }
79     }
80 }
81 document.ondragover = function(e){e.preventDefault();};
82 document.ondrop = function(e){e.preventDefault();}
</script>

```

在本案例的脚本代码中，先创建了一个全局对象 `sendInformation`，并且为该对象设置三个属性——`fileType` 属性、`fileName` 属性与 `content` 属性，其中 `fileType` 的属性值可以被设定为 1、2 或 3，当被上传的文件为图像文件时设定该属性值为 1，当被上传的文件为文本文件时设定该属性值为 2，当被上传的文件为其他文件时设定该属性值为 3。`fileName` 属性代表被拖动文件的文件名（不包括路径），因为在本案中，上传后的文件名与客户端该文件的文件名保持一致，所以需要用一个属性保存被拖动文件的文件名。`content` 属性代表文件内容，当被上传的文件为图像文件时，将该图像文件的二进制数据保存在该属性中；当被上传的文件为文本文件时，将该文本文件的内容保存在该属性中。

本案例的脚本代码中还包括以下两个函数。

#### ❑ init 函数。

先来分析一下此脚本代码中 `init` 函数。当页面打开时将调用该函数，设置文件预览区（id 为 `show` 的 `div` 元素），使其能够接收文件拖放，并且当图像文件或文本文件被拖入时显示被拖入的图像或文本文件的内容。

在 `init` 函数的第 4 行代码中，将代表文件预览区的 id 为 `show` 的 `div` 元素保存到变量 `dest` 中。

在第 5 行到第 14 行的代码中，对文件预览区指定监听两个事件，当有文件被拖动到文件预览区内并在该预览区内移动时（触发 `dragover` 事件）以及拖放结束时（触发 `dragend` 事件），均不执行浏览器的默认动作（在浏览器中打开新窗口并显示文件内容或者直接打开该文件）。

在第 16 行到第 54 行代码中，对文件预览区指定监听一个事件，指定当有文件被拖动到文件预览区并且松开鼠标时（触发 `dragdrop` 事件）所要执行的动作。

在第 17、18 两行代码中，首先指定文件被拖动到文件预览区并且松开鼠标后不执行浏览器的默认动作（在浏览器中打开新窗口并显示文件内容或者直接打开该文件）。

在第 20 行代码中，将代表被拖动文件的对象 `ev.dataTransfer.files[0]` 保存到 `file` 变量中。在第 21 行代码中，使用变量 `reader` 来创建一个 `FileReader` 对象。在第 22 行代码中，将被拖动文件的文件名（不包括路径）保存到 `sendInformation` 对象的 `fileName` 属性中。在第 23 行代码中，将上传按钮保存在变量 `saveButton` 中，在之后对按钮的有效状态进行控制时，将使用该变量进行操作。

在第 25 行到第 35 行的代码中，指定如果被拖动文件是图像文件时所要执行的动作。在第 26 行到第 29 行的代码中，指定读取被拖动的图像文件时将该图像文件作为代表文件预览区的 `div` 元素的背景图像进行显示，同时清除该 `div` 元素中的原有内容（可能是“文件预览区”文字，也可能是已经显示的图像文件、文本文件或者“暂不支持此类文件的预览”文字）。在第 30 行代码中，使用 `FileReader` 对象的 `readAsDataURL` 方法读取图像文件。在第 31 行代码中，使用 Firefox 浏览器中提供的文件对象的 `getAsBinary` 方法读取被拖放图像文件的二进制数据并且将其保存在 `sendInformation` 对象的 `content` 属性中。

在第 33 行与第 34 行代码中，分别将 `sendInformation` 对象的 `fileType` 属性值设定为 1（表示被拖动的文件为图像文件），将上传按钮设定为有效状态。

在第 36 行到第 45 行的代码中，指定如果被拖动文件是文本文件时所要执行的动作。在第 37 行代码中，使用 `FileReader` 对象的 `readAsText` 方法读取文本文件。在第 38 行到第 42 行的代码中，分别指定将文本文件的内容保持原样地显示在文件预览区中（将文本文件内容的两端加上 `pre` 标签，并且将该文本文件的内容指定为文件预览区的 `innerHTML` 属性的属性值），将文本文件的内容设定为 `sendInformation` 对象的 `content` 属性的属性值，同时将文件预览区的背景设定为白色，取消当前有可能被设置为背景的背景图像（因为有可能先拖放图像文件到文件预览区中，在还没有上传该图像文件时立即拖动一个文本文件到文件预览区中）。

在第 43 行与第 44 行的代码中，分别将上传按钮设定为有效状态，将 `sendInformation` 对象的 `fileType` 属性值设定为 2（表示被拖动的文件为文本文件）。

在第 46 行到第 51 行的代码中，指定如果被拖动文件是其他文件时所要执行的动作分别为：在文件预览区中显示“暂不支持此类文件的预览”文字；将文件预览区的背景设定为白色，取消当前有可能被设置为背景的背景图像；设定上传按钮为无效状态；将 `sendInformation` 对象的 `fileType` 属性值设定为 3（表示被拖动的文件为其他文件）。

最后，在第 52 行代码中，将显示“上传成功”的 `div` 元素中的文字清除（因为有可能已

经成功上传了一个文件)。

□ `uploadAndSubmit` 函数。

接下来分析脚本代码中 `uploadAndSubmit` 函数。该函数在用户单击了上传按钮之后被调用。此函数的作用是将表单以及被拖放文件中的内容提交到服务器端文件 `dragAndSaveBack.aspx` 中, 并通过该文件在后端执行文件的上传。在 `uploadAndSubmit` 函数中, 在执行完服务器端的提交后, 程序还将等待服务器端传回的响应, 然后根据上传是否成功来设置上传按钮的文字, 文件预览区中的文字、背景等页面元素的状态。

在第 57 行到第 58 行的代码中, 利用 `xhr` 变量创建一个 `XMLHttpRequest` 对象, 通过该对象来执行 AJAX 上传; 指定 `dragAndSaveBack.aspx` 文件为服务器端后台执行的用来上传文件的脚本文件, 同时将 `sendInformation` 对象的 `fileName` 属性的属性值 (代表被上传的文件名) 和 `fileType` 属性的属性值 (区别上传文件是图像文件、文本文件还是其他文件) 作为参数提交给 `dragAndSaveBack.aspx` 文件。

在第 60 行到第 63 行的代码中, 利用 `XMLHttpRequest` 对象的 `sendAsBinary` 方法或 `send` 方法执行表单的提交, 其中 `sendAsBinary` 方法为在 Firefox 浏览器中自定义的一个方法, 该方法主要用来向服务器端提交二进制数据。如果需要上传的文件为图像文件, 则调用 `sendAsBinary` 方法; 如果需要上传的文件为文本文件, 则调用 `send` 方法。这两个方法使用的参数均为 `sendInformation` 对象的 `content` 属性中所保存的内容, 当被拖放的文件为图像文件时, 该属性中保存图像文件的二进制数据; 当被拖放的文件为文本文件时, 该属性中直接保存文本文件的内容。该参数将 `content` 属性中所保存的内容作为一串数据流 (stream) 提交到服务器端。

在第 65 行到第 77 行的代码中, 指定客户端在收到服务器端传回来的响应后要执行的操作。其中第 67 行表示将服务器端输出的一串文本内容 (`responseText`) 保存到变量 `result` 中, 如果上传成功, 将在服务器端输出的该文本内容中写入“上传成功”; 如果上传失败, 则在该文本内容中写入“上传失败”。第 68 行到第 71 行的代码分别实现在文本预览区中显示文字“文件预览区”, 将文本预览区的背景设定为白色, 取消当前有可能被设置为背景的背景图像, 在上传按钮下部显示服务器端输出的文本内容 (“上传成功”或“上传失败”), 以及将上传按钮设定为无效状态。

第 73 行到第 78 行代码分别实现在服务器完成发送响应后, 如果服务器端传回的文本内容为“上传成功”, 则将上传按钮的文字设定为“继续上传”; 否则将上传按钮的文字设定为“重新上传”。

最后, 在第 81 行与第 82 行的代码中, 设定当页面打开时指定有文件移动到页面上或者在页面上被拖动时不执行页面对拖放处理的默认动作 (在浏览器中打开新窗口并显示文件内容或者直接打开该文件)。

#### 4. ASP.NET 服务器端脚本代码

最后来看一下该案例中的 ASP.NET 服务器端脚本代码, 如代码清单 4-4 所示。

代码清单 4-4 案例中的 ASP.NET 服务器端脚本代码

```

using System;
using System.IO;
using System.Text;
namespace HTML5TEST
{
    public partial class dragAndSaveBack : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            String strResult=String.Empty;
            try
            {
                String fileName = Request.QueryString["fileName"].ToString();
                fileName=Server.UrlDecode(fileName).ToString();
                FileStream fs = new FileStream(fileName, FileMode.Create);
                Stream instream = Request.InputStream;
                Int32 len = Convert.ToInt32(instream.Length);
                if (Request.QueryString["fileType"].ToString().Equals("1"))
                {
                    System.IO.BinaryReader br =
                        new System.IO.BinaryReader(instream);
                    byte[] buffer = br.ReadBytes(len);
                    BinaryWriter w = new BinaryWriter(fs);
                    for (int i = 0; i < buffer.Length; i++)
                        w.Write(buffer[i]);
                    w.Flush();
                    w.Close();
                    fs.Close();
                }
                else
                {
                    int count = 0;
                    byte[] buffer = new byte[1024];
                    StringBuilder builder = new StringBuilder();
                    while ((count = instream.Read(buffer, 0, 1024)) > 0)
                    {
                        builder.Append(Encoding.UTF8.GetString(buffer, 0,
                            count));
                    }
                    StreamWriter w = new StreamWriter(fs,
                        Encoding.GetEncoding("utf-8"));
                    w.WriteLine(builder.ToString());
                    w.Close();
                    fs.Close();
                }

                strResult = "上传成功";
            }
            catch
            {

```

```

        strResult = "上传失败";
    }
    finally
    {
        Response.Write(strResult);
        Response.Flush();
        Response.Close();
    }
}
}
}
}

```

关于 ASP.NET 服务器端代码的讲解，不在本书所介绍的范围之内，因此在这里不做详细介绍，只是概要介绍一下这段代码的功能。同时，如果读者使用的是 JSP 或 PHP 开发语言，也可以将这段代码所完成的功能用 JSP 或 PHP 开发语言实现。

在这段代码中，根据前端提交过来的地址栏参数（Request.QueryString）中的 fileType 参数的参数值，执行相应的写文件操作。如果 fileType 参数值为 1，表示上传文件为图像文件，使用 BinaryWriter 对象，将提交过来的二进制数据（保存在 Request.InputStream 对象中）保存在文件中，文件名为查询字符串中的 fileName 参数的参数值（即客户端该文件的文件名）；如果 fileType 参数值为 2，表示上传文件为文本文件，使用 StreamWriter 对象，将提交过来的文本数据（保存在 Request.InputStream 对象中）保存在文件中，文件名为查询字符串中的 fileName 参数的参数值（即客户端该文件的文件名）。如果保存成功，则将“上传成功”作为服务器端输出的文本数据返回到客户端，否则（保存失败）将“上传失败”作为服务器端输出的文本数据返回到客户端。

## 4.2 案例 12：使用 Canvas API、文件 API 与拖放 API 制作拼图游戏

### 4.2.1 案例概述

本节通过一个综合运用 HTML 5 中 Canvas API、文件 API 与拖放 API 制作拼图游戏的案例，介绍如何使用文件 API 中的 Blob 对象检测用户选择的客户端计算机中文件的类型，如何使用拖放 API 在网页中对元素进行拖动，以及如何将 Canvas API、文件 API 与拖放 API 结合起来制作一个拼图游戏。本案例制作的拼图游戏与目前网络上的拼图游戏均不相同。目前网络上的拼图游戏所使用的图片均保存在服务器中，而且在服务器端必须事先准备好已经被打乱了拼图，如果一幅图片有 25 块拼图，则事先必须准备好这 25 块拼图，然后在网页打开时装入这 25 块拼图，待用户将这 25 块拼图拼成一幅完整的图片后，向用户显示拼图成功的提示信息。而在本案例中，用户可以选择客户端计算机中的任何一幅自己喜欢的图片，单击“创建拼图”按钮后在网页右边的 div 元素中即时生成打乱后的拼图，之后将该 div 元素中的拼图拖动到网页左边的拼图区域中，拼图成功后浏览器向用户显示拼图成功的提示信

息。另外，本案例利用 HTML 5 中文件 API 的 Blob 对象来检测用户选择的文件是否为图像文件，如果不是则显示提示信息，提示用户该文件不是图像文件。

另外，由于本案例中使用了 Firefox 浏览器自定义的一个 `getAsDataURL` 方法来将用户选择的图像绘制在页面上，因此本案例只能运行在 Firefox 浏览器中。

### 4.2.2 页面显示效果

接下来看一下本案例中的页面在浏览器中的显示效果。在打开页面时，页面顶部显示网页标题文字“结合使用 Canvas API、文件 API 与拖放 API 制作拼图游戏”，网页左边显示一个空白的 5 行 5 列的表格，该表格将作为拼图区域使用；网页右边显示一个文字为“选择文件：”的标签，一个供用户选择计算机中图像文件的 `file` 元素，一个文字为“创建拼图”的按钮，以及一个 `div` 元素，在页面打开时该 `div` 元素中显示“请选择文件”文字。创建拼图按钮在该页面打开时为无效状态，如图 4-8 所示。

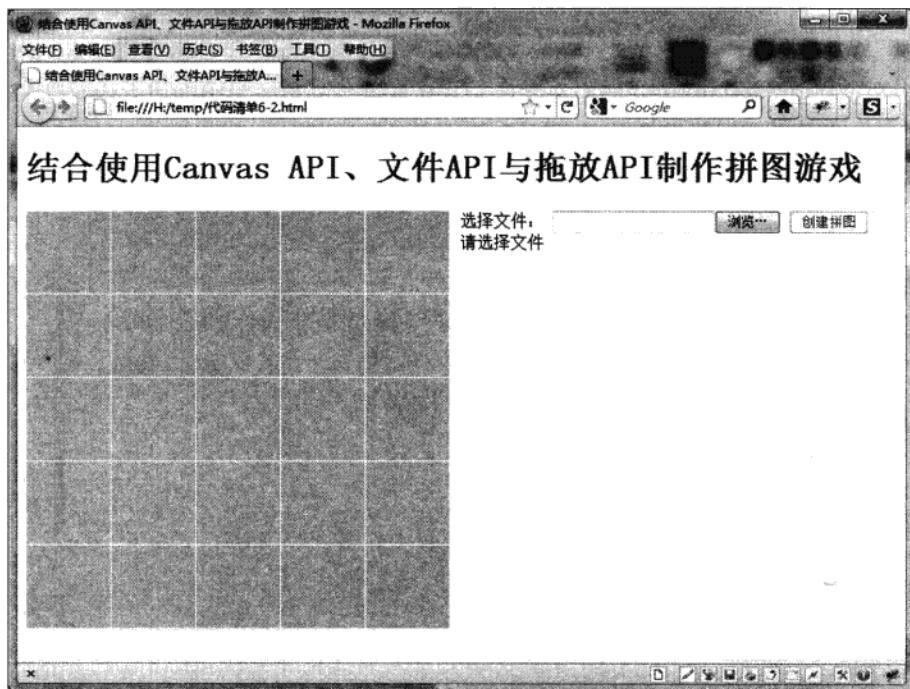


图 4-8 页面打开时的显示效果

在案例页面中，由于对 `file` 元素使用了 `accept` 属性，并且将属性值设定为“`image/*`”，因此用户单击浏览按钮后，在浏览器中打开选择文件对话框，其中默认的文件类型过滤器为“图像文件”，如图 4-9 所示。



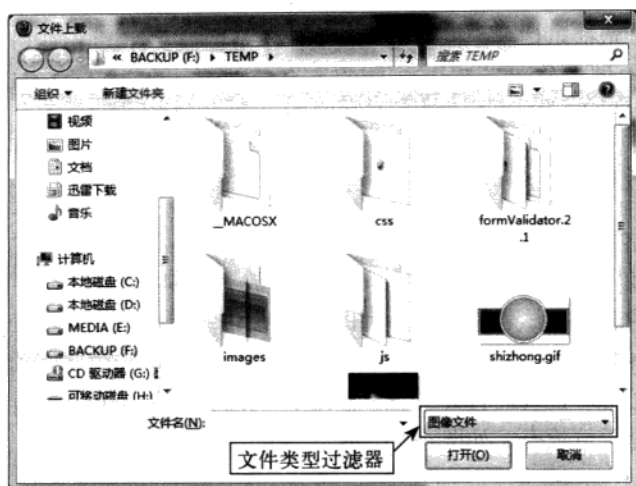


图 4-9 默认的文件类型过滤器为“图像文件”

用户在选择文件对话框中选择图像文件，并且单击打开按钮关闭选择文件对话框后，创建拼图按钮变为有效状态，如图 4-10 所示。

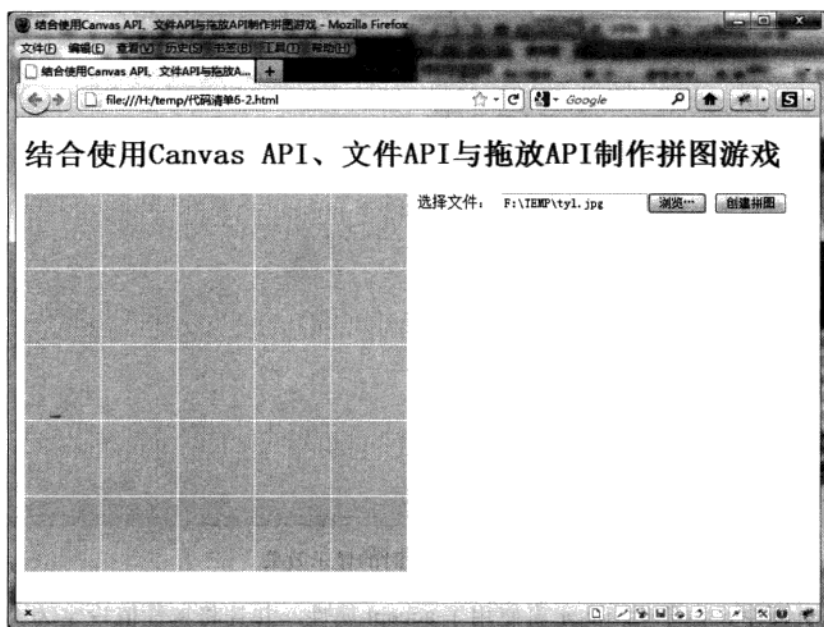


图 4-10 选择图像文件完毕后创建拼图按钮变为有效状态

如果用户选择的文件不是图像文件，则浏览器中弹出提示信息窗口，提示用户选择的文件不是图像文件，如图 4-11 所示。

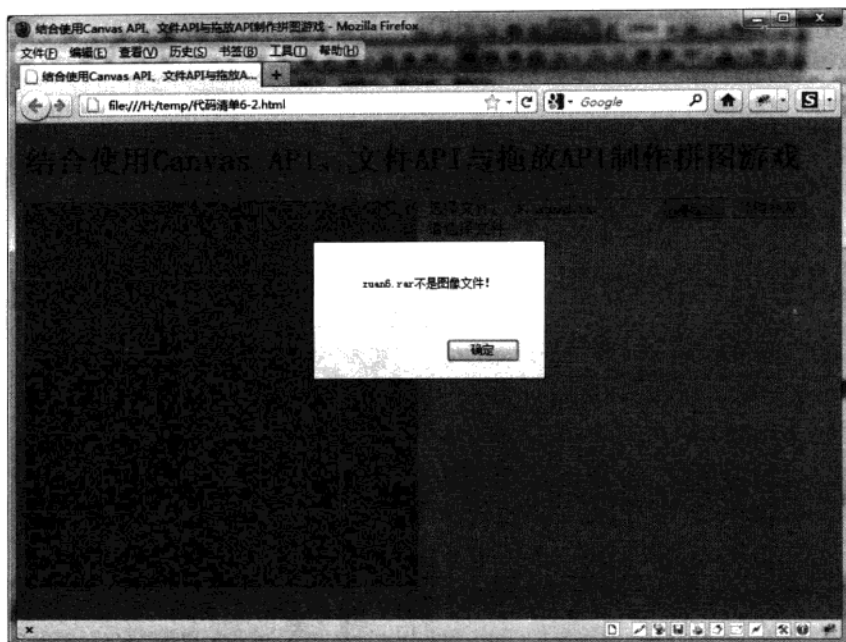


图 4-11 用户选择的不是图像文件时弹出提示信息窗口

如果用户选择的文件不是图像文件，则创建拼图按钮仍然为无效状态，如图 4-12 所示。

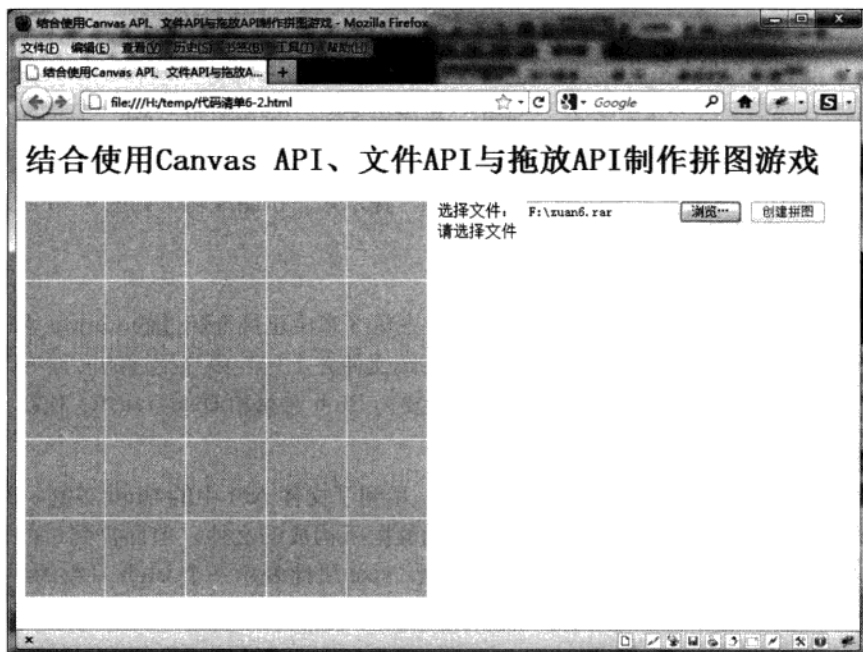


图 4-12 用户选择的文件不是图像文件时创建拼图按钮仍然为无效状态

用户选择图像文件并且单击创建拼图按钮后，将在右边的 div 元素中显示一个 5×5 的表格，该表格中显示将图像文件打乱成 25 块后生成的拼图，如图 4-13 所示。



图 4-13 单击创建拼图按钮后将创建并显示由 25 块组成的拼图

用户进行拼图时，可以将右边 div 元素中的拼图拖动到左边的拼图区域中，如图 4-14 所示。用户也可以将左边拼图区域中的拼图拖回到右边的 div 元素中，如图 4-15 所示。用户拼图成功后，浏览器中将弹出提示信息，提示用户拼图成功，如图 4-16 所示。

### 4.2.3 案例知识点

接下来分析本案例涉及的几个知识点，除了在第 3 章中已经介绍过的 Canvas API 的有关知识点以及“4.1 案例 11：在浏览器中预览客户端文件并上传”中介绍过的文件 API 与拖放 API 的有关知识点之外，本案例用到的知识点主要为 Blob 对象和 DataTransfer 对象。

#### 1. Blob 对象

在检查用户选择的文件是否为图像文件时，用到了文件 API 中的 Blob 对象。Blob 表示二进制原始数据，而 Blob 对象则表示由二进制数据所构成的文件，前面提到过的 file 对象则继承了这个 Blob 对象。Blob 对象有两个属性，size 属性表示一个 Blob 对象的字节长度，type 属性表示 Blob 的 MIME 类型，如果此类型是未知类型，则返回一个空字符串。本案例

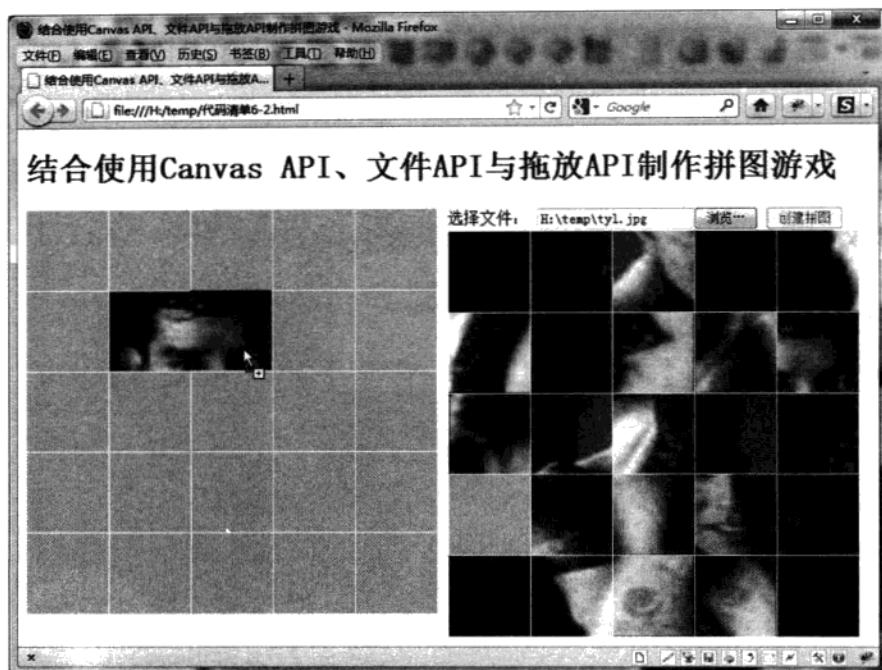


图 4-14 用户将右边 div 元素中的拼图拖动到左边的拼图区域中

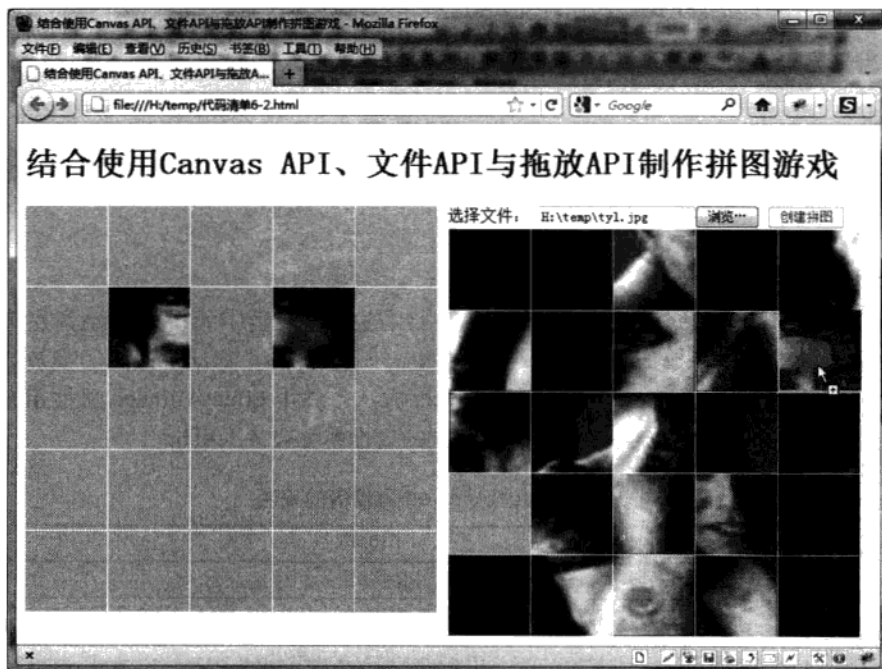


图 4-15 用户将拼图区域中的拼图拖回到右边的 div 元素中

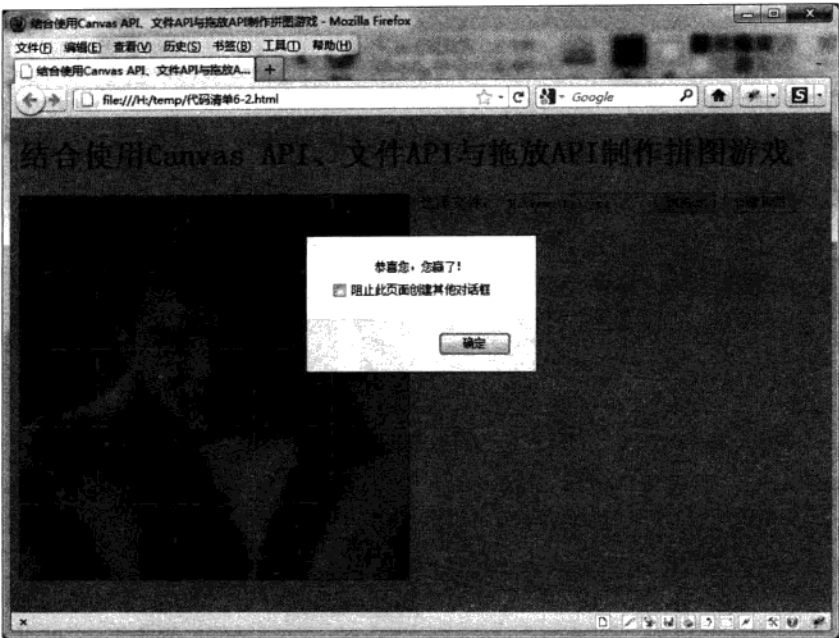


图 4-16 用户拼图成功后浏览器中弹出提示用户拼图成功的提示信息

就利用这个 type 属性来检测用户所选择的文件是否为图像文件，代码如下所示。

```
var file = document.getElementById("file").files[0];
if (!/image\/\w+/.test(file.type))
{
    alert(file.name+" 不是图像文件! ");
    document.getElementById("divShowPic").innerHTML=" 请选择文件 ";
    return;
}
```

2. DataTransfer 对象

另外，本案例通过 HTML 5 中的拖放 API 将右边 div 元素中的拼图拖动到左边的拼图区域中，拖动的对象为一个 canvas 元素（用来显示一块拼图），拖动时用到了拖放 API 中的 DataTransfer 对象的 effectAllowed 属性与 setData 方法，其中 effectAllowed 属性用来指定当元素被拖放时所允许的操作，可以为该属性指定的属性值如表 4-1 所示。

表 4-4 effectAllowed 属性值的说明

属性值	说明
copy	允许将被拖动元素复制到拖动的目标元素中
move	允许将被拖动元素移动到拖动的目标元素中
link	通过拖放操作，将被拖动元素链接到拖动的目标元素中

(续)

属性值	说明
copyLink	将被拖动元素复制或链接到拖动的目标元素中, 根据拖动的目标元素来决定执行复制操作还是链接操作
copyMove	被拖动元素被复制或移动到拖动的目标元素中, 根据拖动的目标元素来决定执行复制操作还是移动操作
linkMove	将被拖动元素链接或移动到拖动的目标元素中, 根据拖动的目标元素来决定执行链接操作还是移动操作
all	允许执行所有拖动操作 (包括复制操作、移动操作与链接操作)
none	不允许执行任何拖动操作
uninitialize	不指定 effectAllowed 属性值。这时将执行浏览器中默认允许的拖动操作, 但是该操作不能通过 effectAllowed 属性值来获取

dropEffect 属性表示实际拖放时的视觉效果, 一般在 ondragover 事件中指定, 允许设定的值为 none、copy、link、move。dropEffect 属性所表示的实际视觉效果必须与 effectAllowed 属性值所表示的允许操作相匹配。规则如下:

- 1) 如果 effectAllowed 属性设定为 none, 则不允许拖放元素。
- 2) 如果 dropEffect 属性设定为 none, 则不允许将元素拖放到目标元素中。
- 3) 如果 effectAllowed 属性设定为 all 或不设定, 则 dropEffect 属性允许被设定为任何值。
- 4) 如果 effectAllowed 属性设定为具体操作, 而 dropEffect 属性也设定了具体视觉效果, 则 dropEffect 属性值必须与 effectAllowed 属性值相匹配, 否则不允许将被拖放元素拖放到目标元素中。

setData 方法用来把要拖动的数据存入 DataTransfer 对象, 该方法的使用如下所示。

```
dt.setData("text/plain", '要存放的数据');
```

setData 方法中的第一个参数为携带数据的数据种类, 第二个参数为要携带的数据。第一个参数的数据种类中只能填入类似 “text/plain” 或 “text/html” 等表示 MIME 类型的文字, 不能填入其他文字。在此参数中可以指定的类型有以下几种:

- text/plain: 文本文字;
- text/html: HTML 文字;
- text/xml: XML 文字;
- text/uri-list: URL 列表, 每个 URL 为一行。

## 4.2.4 代码剖析

接下来具体分析本案例的整个实现过程及实现代码。

### 1. HTML 页面代码

在案例页面的顶部, 使用 h1 元素显示一个标题文字 “综合使用 Canvas API、文件 API 与拖放 API 制作拼图游戏”。在标题文字的下部, 分为左右两个部分, 左边以 table 元素作

为拼图区域；右边的顶部通过一个 file 元素供用户选择图像文件，在 file 元素的右边显示一个创建拼图按钮，file 元素与创建拼图按钮的下部是一个 div 元素，用来显示被打乱的拼图，页面打开时在 div 元素中显示文字“请选择文件”，用户单击创建拼图按钮后在该 div 元素中显示创建好的拼图，拼图数量为 25 块。拼图的创建规则为：将用户选择的图像文件分为 5 行 5 列共 25 块拼图，然后将这些拼图的顺序打乱后显示在 file 元素与创建拼图按钮下部的 div 元素中。另外，在该 div 元素的下部，还有一个 div 元素，当用户选择图像文件之后，会在此 div 元素中显示一个隐藏的 canvas 元素，并且将选择的图像文件中的完整图像绘制在该 canvas 元素中，通过对该完整图像的绘制来取得用户所选择图像的宽度与高度。

案例页面的完整的 HTML 页面代码如代码清单 4-5 所示。

#### 代码清单 4-5 案例页面的完整的 HTML 页面代码

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title>使用 Canvas API、文件 API 与拖放 API 制作拼图游戏 </title>
<style>
... 样式代码部分稍后介绍 ...
</style>
<script type="text/javascript">
... 脚本代码部分稍后介绍 ...
</script>
</head>
<body onload="window_onload()">
<header>
<h1>使用 Canvas API、文件 API 与拖放 API 制作拼图游戏 </h1>
</header>
<div id="divMain">
<div id="divLeft">
<table border="0" cellspacing="1" cellpadding="1" id="leftTable">
<tr>
<td tag="td"></td>
<td tag="td"></td>
<td tag="td"></td>
<td tag="td"></td>
<td tag="td"></td>
</tr>
<tr>
<td tag="td"></td>
<td tag="td"></td>
<td tag="td"></td>
<td tag="td"></td>
<td tag="td"></td>
</tr>
<tr>
<td tag="td"></td>
<td tag="td"></td>
```

```

        <td tag="td"></td>
        <td tag="td"></td>
        <td tag="td"></td>
    </tr>
    <tr>
        <td tag="td"></td>
        <td tag="td"></td>
        <td tag="td"></td>
        <td tag="td"></td>
        <td tag="td"></td>
    </tr>
    <tr>
        <td tag="td"></td>
        <td tag="td"></td>
        <td tag="td"></td>
        <td tag="td"></td>
        <td tag="td"></td>
    </tr>
</table>
</div>
<div id="divRight">
    <div id="divSelectPic">
        选择文件:
        <input type="file" id="file" onchange="selectFile()"
        accept="image/*"/>
        <input type="button" id="createPuzzle" value=" 创建拼图 "
        disabled="disabled"
        onclick="drawTable()" />
    </div>
    <div id="divShowPic">
        请选择文件
    </div>
    <div id="divHiddenPic" />
</div>
</body>

```

---

## 2. 样式代码

接下来分析本案例页面中的样式代码部分，该页面使用了 CSS 3 中的弹性盒布局方式。在标题下方以一个 div 元素作为容器（id 为 divMain），在此元素中放置左右两个 div 元素（id 分别为 divLeft 与 divRight），对 id 为 divMain 的 div 元素使用弹性盒布局的方式使其中的两个 div 元素为左右两列的横向布局方式，样式代码如下。

```

div#divMain{
    width:800px;
    display: -moz-box;
    display: -webkit-box;
    -moz-box-orient:horizontal;
    -webkit-box-orient:horizontal;
}

```



在右边 id 为 divRight 的 div 元素中，从上到下分别使用了三个 div 元素，在第一个 div 元素 (id 为 divSelectPic) 中显示用户选择图像文件所用的 file 元素与创建拼图按钮，在第二个 div 元素 (id 为 divShowPic) 中显示被打乱的拼图，在第三个 div 元素 (id 为 divHiddenPic) 中显示隐藏的 canvas 元素（通过在该 canvas 元素中绘制用户所选择图像来获取该图像的宽度与高度）。对右边 id 为 divRight 的 div 元素使用弹性盒布局的方式使其中的三个 div 元素为垂直方向排列显示，样式代码如下。

```
div#divRight{
    padding-left:10px;
    width:400px;
    display: -moz-box;
    display: -webkit-box;
    -moz-box-orient:vertical;
    -webkit-box-orient:vertical;
}
```

案例页面的完整样式代码如代码清单 4-6 所示。

代码清单 4-6 案例页面的完整样式代码

---

```
<style>
header{
    text-align:center;
    width:800px;
}
div#divMain{
    width:800px;
    display: -moz-box;
    display: -webkit-box;
    -moz-box-orient:horizontal;
    -webkit-box-orient:horizontal;
}
div#divLeft td{
    border-width:1px;
    width:80px;
    height:80px;
    background:#ccc;
}
div#divRight{
    padding-left:10px;
    width:400px;
    display: -moz-box;
    display: -webkit-box;
    -moz-box-orient:vertical;
    -webkit-box-orient:vertical;
}
div#divShowPic{
    border:none;
```

```

        width:400px;
        height:400px;
    }
    td{
        background:#ccc;
        padding:0px;
        margin:0px;
        border:1px;
    }
</style>

```

### 3. JavaScript 脚本代码

接下来分析本案例中的 JavaScript 脚本代码部分，此脚本代码中主要有三个函数：window\_onload 函数、selectFile 函数和 drawTable 函数。

在页面打开时调用 window\_onload 函数对显示画面左边拼图区域的表格中的每一个单元格添加事件进行处理，当有拼图被拖动到表格的每一个单元格中并且松开鼠标时，该拼图能够被放置在单元格中，而当由表格的所有单元格中放置的拼图所拼接起来的完整图像与用户选择的图像文件中的图像完全一致时显示提示信息，提示用户拼图成功。

在用户单击了浏览按钮选择图像文件之后，调用 selectFile 函数将用户选择的图像文件中的图像绘制在画面中隐藏的 canvas 元素中（在创建拼图时以该 canvas 元素中的图像作为拼图的图像来源）。如果用户选择的文件不是图像文件，则显示提示信息，提示用户所选文件不是图像文件。

在用户单击了创建拼图按钮之后，调用 drawTable 函数在 file 元素与创建按钮下方的 div 元素中创建一个 5 行 5 列的表格，每一个单元格中放置一个 canvas 元素，同时将用户选择的图像分为 5 行 5 列共 25 块拼图，并且将这 25 块拼图按照随机顺序绘制在表格中的 25 个 canvas 元素中。

接下来通过代码清单 4-7 来了解本案例中完整的 JavaScript 脚本代码，之后对这个脚本代码进行详细分析。

代码清单 4-7 案例的 JavaScript 脚本代码部分

```

<script type="text/javascript">
1   var image,imageWidth,imageHeight,draggedCanvas;
2   document.ondragover = function(e){e.preventDefault();};
3   document.ondrop = function(e){e.preventDefault();};
4   function selectFile()
5   {
6       var td;
7       for(var i=0;i<5;i++)
8       {
9           for(var j=0;j<5;j++)
10          {
11              td=document.getElementById("leftTable").rows[i].cells[j];

```

```

12         td.innerHTML="";
13     }
14 }
15 var file = document.getElementById("file").files[0];
16 if(!/image\/\w+/.test(file.type))
17 {
18     alert(file.name+" 不是图像文件! ");
19     document.getElementById("divShowPic").innerHTML=" 请选择文件 ";
20     return;
21 }
22 else
23 {
24     document.getElementById("divShowPic").innerHTML="";
25     document.getElementById("divHiddenPic").innerHTML="";
26     canvas=document.createElement("canvas");
27     canvas.style.display="none";
28     document.getElementById("divHiddenPic").appendChild(canvas);
29     var ctx=canvas.getContext('2d');
30     image=new Image();
31     image.src=document.getElementById("file").files[0]
32     .getAsDataURL();
33     image.onload=function()
34     {
35         ctx.drawImage(image,0,0);
36         imageWidth=image.width;
37         imageHeight=image.height;
38     }
39     document.getElementById("createPuzzle").disabled="";
40 }
41 function drawTable()
42 {
43     var canvasArray,canvas,ctx,index,count,div;
44     canvasArray=new Array();
45     div=document.getElementById("divShowPic");
46     table=document.createElement("table");
47     table.setAttribute("border","0");
48     table.setAttribute("cellpadding","0");
49     table.setAttribute("cellspacing","1");
50     picWidth=400;
51     picHeight=400;
52     for(var i=0;i<5;i++)
53     {
54         for(var j=0;j<5;j++)
55         {
56             canvas=document.createElement("canvas");
57             ctx=canvas.getContext('2d');
58             canvas.setAttribute("width",picWidth/5+"px");
59             canvas.setAttribute("height",picHeight/5+"px");
60             canvas.setAttribute("draggable","true");

```

```

61         canvas.style.display="block";
62         canvas.setAttribute("index",i*5+j);
63         canvas.addEventListener("dragstart", function(ev)
64         {
65             draggedCanvas=ev.srcElement||ev.target;
66             var dt = ev.dataTransfer;
67             dt.effectAllowed = 'all';
68             dt.setData("text/plain",draggedCanvas.
               getAttribute("index"));
69         }, false);
70         ctx.fillRect(0,0,canvas.width,canvas.height);
71         ctx.drawImage(image,j*imageWidth/5,i*imageHeight/5,
               imageWidth/5,imageHeight/5,0,0,picWidth/5,picHeight/5);
72         canvasArray.push(canvas);
73     }
74 }
75 count=25;
76 for(var i=0;i<5;i++)
77 {
78     tr=document.createElement("tr");
79     for(var j=0;j<5;j++)
80     {
81         td=document.createElement("td");
82         td.setAttribute("tag","td");
83         td.addEventListener("dragend", function(ev)
84         {
85             ev.preventDefault();
86         }, false);
87         td.addEventListener("drop", function(ev)
88         {
89             var td=ev.srcElement||ev.target;
90             if(td.getAttribute("tag")!=null)
91             {
92                 td.appendChild(draggedCanvas);
93             }
94             ev.preventDefault();
95             ev.stopPropagation();
96         }, false);
97         tr.appendChild(td);
98         index=parseInt(Math.random()*count);
99         td.appendChild(canvasArray[index]);
100         canvasArray.splice(index,1);
101         count-=1;
102     }
103     table.appendChild(tr);
104 }
105 div.appendChild(table);
106 document.getElementById("createPuzzle").disabled="disabled";
107 }
108 function window_onload()

```

```

109  {
110      var td;
111      for(var i=0;i<5;i++)
112      {
113          for(var j=0;j<5;j++)
114          {
115              td=document.getElementById("leftTable").rows[i].cells[j];
116              td.addEventListener("dragend", function(ev)
117              {
118                  ev.preventDefault();
119              }, false);
120              td.addEventListener("drop", function(ev)
121              {
122                  var td=ev.srcElement||ev.target;
123                  if(td.getAttribute("tag")!=null)
124                  {
125                      td.appendChild(draggedCanvas);
126                  }
127                  var allHaveFlag=true;
128                  var indexStr="";
129                  for(var i=0;i<5;i++)
130                  {
131                      for(var j=0;j<5;j++)
132                      {
133                          canvas=document.getElementById("leftTable").
134                              rows[i].cells[j].children[0];
135                          if(canvas==null)
136                          {
137                              allHaveFlag=false;
138                              indexStr="";
139                              break;
140                          }
141                          else
142                          {
143                              indexStr+=canvas.getAttribute("index");
144                          }
145                      }
146                  }
147                  var allStr="0123456789101112131415161718192021222324";
148                  if(allHaveFlag&&indexStr== allStr)
149                      alert("恭喜您,您赢了!");
150                  ev.preventDefault();
151                  ev.stopPropagation();
152              }, false);
153          }
154      }
</script>

```

脚本代码的第一行定义了以下几个全局变量:

#### ❑ 全局变量 image。

变量 image 代表 JavaScript 中的一个 Image 对象。在此对象中装载用户所选择的图像文件中的图像，在绘制 25 块拼图时，将以此 Image 中装载的图像作为绘制拼图的源。

#### ❑ 全局变量 imageWidth 和全局变量 imageHeight。

变量 imageWidth 与变量 imageHeight 分别代表用户选择的图像的宽度与高度。在绘制 25 块拼图时要通过 imageWidth/5 和 imageHeight/5 来计算每一块拼图中所绘制图像的宽度与高度（请注意，不是每一个拼图的宽度与高度，而是拼图中所绘制图像的宽度与高度。例如将宽 600，高 600 的图像绘制在宽 100，高 100 的 canvas 元素中，此时 imageWidth=600，imageHeight=600，每一块拼图的宽度与高度都为  $600/5=120$ ，然后将其缩小绘制在宽 100，高 100 的 canvas 元素中）。

#### ❑ 全局变量 draggedCanvas。

变量 draggedCanvas 代表被拖动的拼图（canvas 元素）。当拖动拼图到拼图区域的单元格中并松开鼠标时，将会把此拼图（canvas 元素）放置在单元格中。将被拖动的拼图（canvas 元素）放置在单元格中的代码如下所示。

```
td.appendChild(draggedCanvas);
```

脚本代码的第 2 行、第 3 行设定整个页面为不执行默认处理（即拒绝被拖放的处理）。

下面介绍一下脚本代码中的 3 个函数。

#### ❑ selectFile 函数。

脚本中第 4 行到第 40 行的代码为 selectFile 函数，当用户单击浏览按钮选择图像文件时将调用该函数。

第 6 行到第 14 行的代码表示将左边拼图区域中当前放置的拼图全部删除。因为用户可能已经选择了一幅图像文件，创建了 25 块拼图，并将部分拼图或全部拼图拖动到拼图区域，所以再次单击浏览按钮重新选择图像文件后应该将拼图区域中的拼图删除。

在第 15 行代码中，使用 file 变量来引用用户所选择的文件。第 16 到第 21 行的代码表示，如果用户选择的文件不是图像文件则显示提示信息，提示用户所选择文件不是图像文件，并且在 file 元素与创建按钮下部的 div 元素中只显示文字“请选择文件”。因为用户可能已经选择了一幅图像文件，并且单击创建拼图按钮在此 div 元素中显示 25 块拼图之后重新选择图像文件，所以应该在用户重新选择图像文件之后将已经创建并显示的上一幅图像的全部拼图删除。

第 22 行到第 39 行的代码表示用户选择图像文件之后所执行的处理，下面介绍这些处理。

第 24 行代码表示将 file 元素与创建按钮下部的 div 元素中当前显示的拼图或文字全部清除。如果是第一次单击浏览按钮，则会将该 div 元素中的“请选择文件”文字删除。如果用户已经选择了图像文件，则单击创建拼图按钮在下部的 div 元素中绘制拼图之后再次单击浏览按钮重新选择图像文件时，将下部 div 元素中绘制的拼图删除。

第 25 行代码表示将位于右边 div 元素底部的 div 元素 (id 为 divHiddenPic) 中的 canvas 元素删除, 因为用户有可能已经单击过一次浏览按钮进行图像文件的选择, 这时会在底部 div 元素中添加一个隐藏的 canvas 元素, 并且在该 canvas 元素中绘制用户所选择文件中的图像, 所以在用户第二次单击浏览按钮进行图像文件的选择时, 应该删除上一次动态添加的 canvas 元素。

第 26 行到第 28 行的代码表示创建一个隐藏的 canvas 元素, 并且向位于右边 div 元素底部的 div 元素 (id 为 divHiddenPic) 中添加该 canvas 元素。

第 29 行代码表示取得该 canvas 元素的图形上下文对象并将其保存在变量 ctx 中。

第 30 行到第 31 行的代码表示创建一个 JavaScript 的 Image 对象并将其保存在 image 中, 同时将该 Image 对象的图像来源设定为用户所选择的图像文件。

第 32 行到第 37 行的代码表示在向 Image 对象中装载图像时将该图像绘制在隐藏的 canvas 元素中, 同时获取该图像的高度与宽度并保存在变量 imageWidth 与变量 imageHeight 中。

第 38 行代码表示将创建拼图按钮从无效状态改变为有效状态。

□ drawTable 函数。

第 41 行到第 107 行的代码为 drawTable 函数。当用户单击创建拼图按钮后将调用此函数在 file 元素与创建按钮下方的 div 元素中显示 25 块拼图。

第 43 行代码定义了本函数中使用到的几个变量。其中变量 canvas 代表一个 canvas 元素, 以该元素充当每一块拼图; 变量 ctx 代表每一个 canvas 元素的图形上下文对象; 变量 canvasArray 代表一个由 25 个 canvas 元素构成的数组; 变量 index 表示显示每一块拼图的 canvas 元素在该数组中的序号; count 变量表示数组的大小, 在函数中将该变量赋值为 25; 变量 div 代表一个 div 元素。

第 44 行代码创建了一个 JavaScript 的数组, 并且将其保存在变量 canvasArray 中。

在第 45 行代码中使用变量 div 来引用页面中 file 元素与创建按钮下部的 div 元素。

第 46 行到第 49 行的代码创建了一个 table 元素, 并且对该 table 元素设置适当的样式。后面的代码将把该 table 元素设置为一个 5 行 5 列的表格, 每个单元格中放置一个用来显示拼图的 canvas 元素。

第 50 行到第 51 行的代码将变量 picWidth 与变量 picHeight 的变量值都设为 400, 每一个 canvas 元素的宽度与高度将被设置为 picWidth/5 与 picHeight/5 (即 80)。

第 52 行到第 74 行的代码将用户选择的图像分为 5 行 5 列共 25 个拼图, 然后将每一块拼图绘制在一个动态创建的 canvas 元素中, 将每一个 canvas 元素的宽度与高度设置为 picWidth/5 与 picHeight/5 (即 80), 设置每一个 canvas 元素为可以被拖放的元素, 每一个 canvas 元素的 display 属性的属性值为 block。同时对每一个 canvas 元素添加一个 dragstart 事件的监听, 指定当此元素被拖动时将此元素保存在变量 draggedCanvas 中, 这样, 当该 canvas 元素被拖动到代表拼图区域的 table 表格的每一个单元格中时, 松开鼠标即可使用 “td.appendChild(draggedCanvas);” 语句将该 canvas 元素 (即拼图) 放置在目标单元格中。第 67 行代码与第 68 行代码表示允许所有拖放操作, 并且将被拖动的 canvas 元素在 canvas

元素数组中的序号作为拖放数据保存在 `dataTransfer` 对象中，虽然该拖放数据没有真正起到什么作用，但是如果 `dataTransfer` 对象中没有被拖放数据，则该元素将不能被拖动。在第 70 行到第 71 行的代码中，将分为 5 行 5 列后的每一小块图像绘制在一个 `canvas` 元素中。第 72 行代码表示将每一个 `canvas` 元素（共 5 行 5 列 25 个 `canvas` 元素）依次放置在 `canvas` 元素数组中。

在第 75 行代码中，将代表 `canvas` 元素数组大小的 `count` 变量值设定为 25。

在第 76 行到 105 行的代码中，第 46 行代码将动态创建的 `table` 元素设定为一个 5 行 5 列的表格，并且动态创建该表格的每一行每一列，同时将每一个单元格 `td` 元素的 `tag` 属性的属性值设定为 `td`，这是为了防止用户在把拼图区域中的一块拼图拖动到表格中的某一个单元格并放下之后，再把拼图区域中的另一块拼图拖动到该单元格之中，从而使同一个单元格中叠放两块拼图。在第 83 行到第 96 行的代码中，对该表格添加对 `dragend` 事件与 `drop` 事件的监听，并且设定当有作为拼图的 `canvas` 元素从拼图区域被拖动到表格中的单元格并松开鼠标时，使用“`td.appendChild(draggedCanvas);`”语句把该 `canvas` 元素放置在单元格中。但是，在单元格中已经放置了一块拼图（`canvas` 元素）之后，再从拼图区域中拖动一块拼图到这个动态创建的表格的单元格中时，仍然会触发此单元格的 `drop` 事件。这时第 89 行中的 `ev.srcElement||ev.target`（拖动的目标元素）将指向第一个被拖动到此单元格中的 `canvas` 元素，因此如果直接使用第 92 行中的“`td.appendChild(draggedCanvas);`”语句，将把第二块拼图（`canvas` 元素）拖动到第一个被拖动到此单元格中的 `canvas` 元素中。为了防止这种情况出现，把此表格中所有 `td` 元素的 `tag` 属性值设定为 `td`，然后在第 92 行的“`td.appendChild(draggedCanvas);`”语句之前，添加代码来判断拖动的目标元素的 `tag` 属性值是否为 `td`，如果是则表示目标元素为 `td` 元素（单元格），在该单元格中添加被拖动的 `canvas` 元素；如果目标元素的 `tag` 属性值不为 `td`，则表示目标元素为 `canvas` 元素，拒绝将被拖动的 `canvas` 元素添加到目标 `canvas` 元素中。

`drawTable` 函数中的最后两行（第 105 行与第 106 行）代码表示将动态创建的放置拼图的表格添加到 `file` 元素与创建拼图按钮下部的 `div` 元素中，然后将创建拼图按钮设置为无效状态。

#### ❑ `window_onload` 函数

第 108 行到第 154 行的代码为 `window_onload` 函数的代码。当页面打开时将调用此函数，对代表拼图区域的表格中的每一个单元格进行设置，当有拼图被拖动到此单元格中并松开鼠标时，将该拼图放置在此单元格中。当所有拼图都被拖动到拼图区域，并且拼接起的完整图像与用户选择的图像文件中的图像完全一致时，显示提示信息，提示用户拼图成功。

第 116 行到第 119 行的代码表示对代表拼图区域的表格中的每一个单元格监听 `dragend` 事件。当拖动结束时将触发该事件，在该事件中设置单元格不执行浏览器指定的默认处理（不接受其他元素被拖动到该元素中的处理）。

第 120 行到第 151 行的代码表示对拼图区域的表格中的每一个单元格监听 `drop` 事件。当有其他元素被拖动到该单元格中时，松开鼠标将触发该事件。在该事件中，使用一个名为

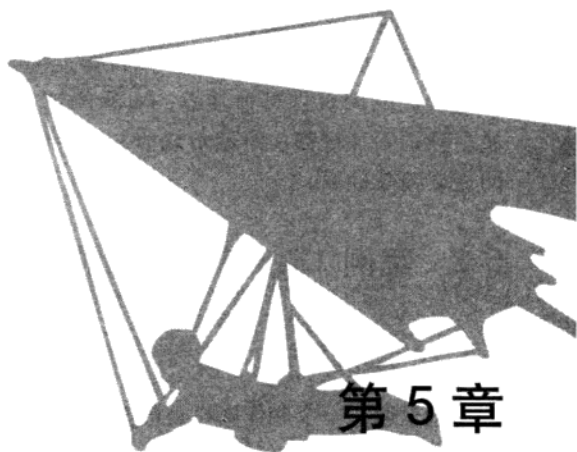


allHaveFlag 的布尔型变量来区别是否所有单元格中都被放入了拼图 (canvas) 元素, 使用一个 indexStr 来保存表格中所有 canvas 元素的编号 (即 canvas 元素的 index 属性的属性值), 同时对表格进行遍历。在遍历的过程中, 将每一个单元格中的 canvas 元素保存在一个 canvas 变量中, 当有一个单元格中没有放入 canvas 元素时, 则将 canvas 元素设置为 NULL, 如果 canvas 元素被设置为 NULL, 则立即中止遍历过程, 如果每一个单元格中都放入了 canvas 元素, 则判断由这些 canvas 元素的编号连接起来生成的 indexStr 字符串是否与一个从 1 到 25 的数字连接起来生成的字符串完全相等, 如果相等则表示这 25 块拼图连接起来的图像与用户选择的图像文件中的图像吻合, 浏览器中显示提示信息, 提示用户拼图成功。

### 4.3 本章小结

本章通过两个案例来详细阐述 HTML 5 中的文件 API 与拖放 API 的基础知识以及使用方法, 力求使读者通过阅读本章内容, 对这两个 API 有一个比较全面的认识, 能够将这两个 API 应用在自己的 Web 网站或 Web 应用程序的开发过程中。

下一章将通过几个案例来具体讲解 HTML 5 中 video 元素与 audio 元素的基础知识及使用方法。读者通过对下一章内容的阅读来了解在 HTML 5 中不通过插件如何在网页上播放视频文件和音频文件, 同时也将了解到如何制作属于自己的网页媒体播放器。



## 第 5 章

# 多媒体播放

### 本章内容

- ☐ 案例 13: 打造自己的网页视频播放器
- ☐ 案例 14: 对视频使用实时回放功能
- ☐ 案例 15: 对视频使用截图功能
- ☐ 案例 16: 打造自己的网页音频播放器
- ☐ 本章小结

本章通过 4 个案例向读者展示如何在浏览器中使用 HTML5 中的 video 元素与 audio 元素，如何在网页中不使用插件即可播放视频文件与音频文件。本章的第一个案例将介绍如何打造自己的网页视频播放器，本章的第二个案例与第三个案例将对本章第一个案例进行扩展，分别添加视频实时回放功能与实时截图功能。本章的第四个案例将介绍如何打造自己的网页音频播放器。

## 5.1 案例 13：打造自己的网页视频播放器

### 5.1.1 案例概述

这里通过一个制作网页视频播放器的案例向读者介绍如何使用 HTML 5 中的 video 元素来播放视频文件，如何利用 video 元素的各种属性、事件与方法来定制属于自己的网页视频播放器。虽然目前很多主流浏览器都针对 video 元素提供了一个视频播放控件（在代码中可以通过使用 video 元素的 controls 属性来显示该控件），但是很多时候这个默认的播放控件并不能满足开发者或用户的某些特定需求，这就需要通过自己定义视频播放器来满足这种需求。

### 5.1.2 页面显示效果

首先来看一下本案例页面在浏览器中的显示效果。在本案例页面中，放置了一个 video 元素来播放视频，不指定该 video 元素的 controls 属性以隐藏浏览器自己提供的播放控件，在视频下部使用一个 HTML5 中的 progress 元素作为播放进度条，在进度条下部显示几个按钮，按钮文字分别为“播放”、“加速播放”、“减速播放”、“慢动作”、“静音”、“增大音量”和“降低音量”。

在一个不支持 video 元素的浏览器中，页面的显示效果如图 5-1 所示。



图 5-1 案例页面在不支持 video 元素的浏览器中的显示效果

在一个支持 video 元素的浏览器中，页面打开时将直接播放页面中临时指定的播放文件。在实际应用本案例的时候，读者可以通过编程的手段来动态指定播放文件（如在视频播放网站中让用户选择所要播放的文件）。在一个不支持 progress 元素的浏览器中，使用 div 元素并通过对 div 元素指定样式的方法来模仿 progress 元素，以呈现一个播放进度条的效果，

如图 5-2 所示。



图 5-2 案例页面在不支持 progress 元素的浏览器中的显示效果

当鼠标指针移动到进度条（div 元素）上时，进度条中将显示视频总的播放时间与当前已播放时间的文字，如图 5-3 所示。



图 5-3 当鼠标指针移动到进度条（div 元素）上时，进度条中将显示播放时间

在支持 progress 元素的浏览器中，案例程序将直接使用 progress 元素来显示进度条，如图 5-4 所示。



图 5-4 案例页面在支持 progress 元素的浏览器中的显示效果

当鼠标指针移动到进度条（progress 元素）上时，进度条中将显示视频总的播放时间与当前已播放时间的文字，如图 5-5 所示。



图 5-5 当鼠标指针移动到进度条（progress 元素）上时，进度条中将显示播放时间

单击暂停按钮后，页面中所有元素处于无效状态，如图 5-6 所示。



图 5-6 单击暂停按钮后，页面中所有元素处于无效状态

### 5.1.3 案例知识点

在详细分析页面代码之前，先介绍一下本案例所使用到的 video 元素的属性、方法与事件，video 元素的完整说明请参阅笔者所著《HTML 5 与 CSS 3 权威指南》。

#### □ Src 属性。

在该属性中指定播放视频的 URL 地址。

#### □ width 属性与 height 属性（video 元素独有属性）。

在该属性中指定视频的宽度与高度（以像素为单位）。这两个属性的使用方法如下所示。

```
<video src="video.webm" width="500" height="500"></video>
```

本案例只使用 width 来指定视频宽度，通过所装载视频的宽度与高度的比例来自动调整 video 元素的高度，代码如下所示。

```
<video id="video" src="video.webm" width="800">
  您的浏览器不支持 video 元素
</video>
```

在以上这段代码中，“您的浏览器不支持 video 元素”为在不支持 video 元素的浏览器中

所显示的文字。

#### ❑ error 属性。

在读取视频的过程中，在正常情况下，video 元素的 error 属性为 null，但是任何时候只要出现错误，error 属性将返回一个 MediaError 对象，该对象的 code 返回对应的错误状态。错误状态共有 4 个可能值，如下所示。

- MEDIA\_ERR\_ABORTED (数字值为 1): 媒体数据的下载过程由于用户的操作而被中止。
- MEDIA\_ERR\_NETWORK (数字值为 2): 确认媒体资源可用，但是在下载时出现网络错误，媒体数据的下载过程被中止。
- MEDIA\_ERR\_DECODE (数字值为 3): 确认媒体资源可用，但是解码时发生错误。
- MEDIA\_ERR\_SRC\_NOT\_SUPPORTED (数字值为 4): 媒体资源不可用或媒体格式不被支持。

error 属性为只读属性。

#### ❑ currentTime 属性与 duration 属性。

可以通过 video 元素的 currentTime 属性来读取视频的当前播放位置，也可以通过修改 currentTime 属性来修改当前播放位置。

可以使用 video 元素的 duration 属性来读取视频总的播放时间。

两个属性的值均为时间，单位为秒，currentTime 为可读写属性，duration 属性为只读属性。

#### ❑ played 属性、paused 属性和 ended 属性。

可以通过 video 元素的 played 属性来返回一个 TimeRanges 对象，从该对象中可以读取视频的已播放部分的时间段。开始时间为已播放部分的开始时间，结束时间为已播放部分的结束时间。

可以通过 video 元素的 paused 属性来返回一个布尔值，表示视频是否处于暂停播放中，true 表示视频暂停播放，false 表示视频正在播放。

可以通过 video 元素的 end 属性来返回一个布尔值，表示视频是否播放完毕，true 表示视频播放完毕，false 表示还没有播放完毕。

这三个属性均为只读属性。

#### ❑ playbackRate 属性。

可以通过 video 元素的 playbackRate 属性读取或修改视频当前的播放速率。

#### ❑ volume 属性与 muted 属性。

可以通过 video 元素的 volume 属性读取或修改视频的播放音量，范围为 0 到 1，0 为静音，1 为最大音量。

可以通过 video 元素的 muted 属性读取或修改视频的静音状态，该属性值为布尔值，true 表示处于静音状态，false 表示处于非静音状态。

#### ❑ canPlayType 属性。

可以简单使用 video 元素的 canPlayType 属性来检查浏览器是否支持 video 元素，是否

支持某种格式的视频文件。

❑ play 方法。

使用 play 方法来播放媒体，自动将元素的 paused 属性的属性值变为 false。

❑ pause 方法。

使用 pause 方法来暂停播放，自动将元素的 paused 属性的属性值变为 true。

❑ loadedmetadata 事件。

当浏览器获取完播放视频的时间长度和字节数时将触发该事件，该事件的触发意味着 video 元素中指定的视频文件能被浏览器正常播放。

❑ ended 事件。

当视频文件播放结束或停止时将触发该事件。

❑ play 事件。

当使用 video 元素的 play 方法播放视频时将触发该事件，或者视频数据被下载完毕且 video 元素被设为 autoplay(自动播放)属性后自动播放视频时将触发该事件。

❑ pause 事件。

当执行了 video 元素的 pause 方法暂停视频的播放时将触发该事件。

❑ timeupdate 事件。

当前播放位置被改变时将触发该事件。播放位置的改变可能是视频播放过程中的自然改变，也可能是人为改变，或者是由于播放不能连续而发生的跳变。

❑ error 事件。

获取视频数据过程中出错时将触发该事件。

### 5.1.4 代码剖析

接下来具体分析本案例的整个实现过程及实现代码。

#### 1. HTML 页面代码与样式代码

本案例的 HTML 页面代码部分比较简单，首先使用一个 video 元素来播放视频，在不支持 video 元素的浏览器中将直接显示文字“您的浏览器不支持 video 元素”。在浏览器的下部使用一个 progress 元素来作为播放进度条，在 progress 元素的内部使用一个 div 元素，在不支持 progress 元素的浏览器中将使用该 div 元素来模仿进度条。在 progress 元素的下部放置一些按钮，文字分别为“播放”、“加速播放”、“减速播放”、“慢动作”、“静音”、“增大音量”和“降低音量”。页面中还使用一个活动的 div 元素，该 div 元素的 id 为“showTime”，当鼠标移动到 progress 元素或模仿 progress 元素的 div 元素上时，把该 div 元素定位在鼠标指针处，并且在该 div 元素中显示视频总的播放时间与当前已播放部分的播放时间。

本案例的 HTML 页面代码与样式代码如代码清单 5-1 所示。



代码清单 5-1 本案例的 HTML 页面代码与样式代码

---

```

<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8"/>
<title> 打造你自己的视频播放器 </title>
<style>
div#buttonDiv{
    width:800px;
    display:none;
}
progress#playPercent{
    width:800px;
}
div#progress{
    width:800px;
    height:20px;
    left:0px;
    background:#ccc;
}
div#progressValue{
    width:0%;
    height:20px;
    background:green;
    cursor:default;
}
div#showTime{
    z-index:2;
    display:none;
    position:absolute;
    font-size:10px;
}
</style>
</head>
<body>
<video id="video" src="video.webm" width="800">
您的浏览器不支持 video 元素
</video>
<div id="buttonDiv">
<progress id="playPercent" max=100>
    <div id="progress">
        <div id="progressValue"></div>
    </div>
</progress>
<button id="btnPlay" onclick="PlayOrPause()" disabled/> 播放 </button>
<button id="btnSpeedUp" onclick="SpeedUp()" disabled/> 加速播放 </button>
<button id="btnSpeedDown" onclick="SpeedDown()" disabled/> 减速播放 </button>
<button id="btnSlowPlay" onclick="SlowPlay()" disabled/> 慢动作 </button>
<button id="btnMute" onclick="setMute()" disabled/> 静音 </button>

```

```

<button id="btnVolumeUp" onclick="VolumeUp()" disabled/>增大音量</button>
<button id="btnVolumeDown" onclick="VolumeDown()" disabled/>
降低音量</button>
</div>
<div id="showTime">
</div>
</body>
<script type="text/javascript">
... 脚本代码稍后介绍 ...
</script>
</html>

```

---

## 2. JavaScript 脚本代码

接下来在代码清单 5-2 中看一下本案例的完整的 JavaScript 脚本代码，稍后将对这个 JavaScript 脚本代码进行详细介绍。

代码清单 5-2 本案例的完整的 JavaScript 脚本代码

---

```

<script type="text/javascript">
var speed=1;
var volume=1;
var muted=false;
var video = document.getElementById("video");
var showTime=document.getElementById("showTime");
if(video.canPlayType)
{
    video.addEventListener('loadedmetadata',loadedmetadata,false);
    video.addEventListener('ended',videoEnded,false);
    video.addEventListener('play',videoPlay,false);
    video.addEventListener('pause',videoPause,false);
    video.addEventListener('timeupdate',updateProgress,false);
    video.addEventListener("error",catchError,false);
    progress=document.getElementById("progress");
    progress.onmouseover=progress_mouseover;
    progress.onmouseout=progress_mouseout;
    progress.onclick=progress_click;
    playPercent=document.getElementById("playPercent");
    playPercent.onmouseover=progress_mouseover;
    playPercent.onmouseout=progress_mouseout;
    playPercent.onclick=playPercent_click;
}
function catchError()
{
    var error = video.error;
    switch(error.code)
    {
        case 1:
            alert("视频的下载过程被中止。");
            break;

```

```

        case 2:
            alert("网络发生故障, 视频的下载过程被中止。");
            break;
        case 3:
            alert("解码失败。");
            break;
        case 4:
            alert("媒体资源不可用或媒体格式不被支持。");
            break;
    }
}

function loadedmetadata()
{
    var btnPlay=document.getElementById("btnPlay");
    btnPlay.innerHTML="暂停";
    btnPlay.disabled="";
    video.play();
    var buttonDiv=document.getElementById("buttonDiv");
    buttonDiv.style.display="block";
}

function PlayOrPause()
{
    if(video.paused)
    {
        video.play();
        video.playbackRate=speed;
        video.muted=muted;
        video.volume=volume;
    }
    else
        video.pause();
}

function videoEnded(ev)
{
    video.currentTime=0;
    this.pause();
}

function videoPlay(ev)
{
    var btnPlay=document.getElementById("btnPlay");
    btnPlay.innerHTML="暂停";
    document.getElementById("btnSpeedUp").disabled="";
    document.getElementById("btnSpeedDown").disabled="";
    document.getElementById("btnSlowPlay").disabled="";
    document.getElementById("btnMute").disabled="";
    document.getElementById("btnVolumeUp").disabled="";
    document.getElementById("btnVolumeDown").disabled="";
}

```

```

function videoPause(ev)
{
    var btnPlay=document.getElementById("btnPlay");
    btnPlay.innerHTML=" 播放 ";
    document.getElementById("btnSpeedUp").disabled="disabled";
    document.getElementById("btnSpeedDown").disabled="disabled";
    document.getElementById("btnSlowPlay").disabled="disabled";
    document.getElementById("btnMute").disabled="disabled";
    document.getElementById("btnVolumeUp").disabled="disabled";
    document.getElementById("btnVolumeDown").disabled="disabled";
}
function updateProgress()
{
    var value=Math.round((Math.floor(video.currentTime)
/Math.floor(video.duration))*100,0);
    var progress = document.getElementById('playPercent');
    progress.value = value;
    var progressValue=document.getElementById("progressValue");
    progressValue.style.width = value+"%";
    showTime.innerHTML=calcTime(Math.floor(video.currentTime)
+'/' +calcTime(Math.floor(video.duration)));
}
function calcTime(time)
{
    var hour;
    var minute;
    var second;
    hour=String(parseInt(time/3600,10));
    if (hour.length == 1)    hour    = '0' + hour;
    minute=String(parseInt((time%3600)/60,10));
    if (minute.length == 1)    minute    = '0' + minute;
    second=String(parseInt(time%60,10));
    if (second.length == 1)    second    = '0' + second;
    return hour+":"+minute+":"+second;
}
function progress_mouseover(ev)
{
    showTime.style.display="inline";
    showTime.style.left=ev.pageX+5+"px";
    showTime.style.top=ev.pageY+5+"px";
    showTime.innerHTML=calcTime(Math.floor(video.currentTime)
+'/' +calcTime(Math.floor(video.duration)));
    ev.stopPropagation();
}
function progress_mouseout(ev)
{
    showTime.style.display="none";
    showTime.innerHTML="";
}
function playPercent_click(evt)

```

```

{
    if (evt.offsetX)
    {
        playPercent=document.getElementById("playPercent");
        video.currentTime = video.duration * (evt.offsetX / playPercent.clientWidth);
    }
}
function progress_click(evt)
{
    progress=document.getElementById("progress");
    if (evt.offsetX)
        video.currentTime = video.duration * (evt.offsetX /progress.clientWidth);
    else
        video.currentTime = video.duration * (evt.clientX /progress.clientWidth);
}
function SpeedUp()
{
    video.playbackRate+=1;
    speed=video.playbackRate;
}
function SpeedDown()
{
    video.playbackRate-=1;
    if (video.playbackRate<0)
        video.playbackRate=0;
    speed=video.playbackRate;
}
function SlowPlay()
{
    var btnSlowPlay=document.getElementById("btnSlowPlay");
    if (btnSlowPlay.innerHTML==" 慢动作 ")
    {
        video.playbackRate=0.5;
        btnSlowPlay.innerHTML=" 正常速度 ";
        document.getElementById("btnSpeedUp").disabled="disabled";
        document.getElementById("btnSpeedDown").disabled="disabled";
    }
    else
    {
        video.playbackRate=1;
        btnSlowPlay.innerHTML=" 慢动作 ";
        document.getElementById("btnSpeedUp").disabled="";
        document.getElementById("btnSpeedDown").disabled="";
    }
    speed=video.playbackRate;
}
function setMute()
{
    if (!video.muted)
    {

```

```

        video.muted=true;
        document.getElementById("btnMute").innerHTML=" 取消静音 ";
    }
    else
    {
        video.muted=false;
        document.getElementById("btnMute").innerHTML=" 静音 ";
    }
    muted=video.muted;
}
function VolumeUp()
{
    if(video.volume<1)
        video.volume+=0.1;
    volume=video.volume;
}
function VolumeDown()
{
    if(video.volume>0)
        video.volume-=0.1;
    volume=video.volume;
}
</script>

```

在这段脚本代码的开始处，定义了本案例中使用的几个全局变量，这些全局变量的含义如下。

- speed: 视频的播放速率。
- volume: 播放视频时的音量。
- muted: 是否使用静音。
- video: 页面中的 video 元素。
- showTime: 页面中用来显示视频总的播放时间与当前已播放时间的 div 元素。

接下来指定在页面打开时直接执行的一些 JavaScript 脚本代码。

在页面打开时，使用 video 元素的 canPlayType 来检测浏览器是否支持 video 元素，如果不支持就不执行任何代码，这时浏览器中将只显示一段“您的浏览器不支持 video 元素”文字。

如果浏览器支持 video 元素，将对 video 元素监听 loadedmetadata 事件（当浏览器获取完播放视频的时间长度和字节数时将触发该事件），指定 loadedmetadata 事件触发时调用 loadedmetadata 函数；然后对 video 元素监听 ended 事件（当视频播放完毕或停止时将触发该事件），指定 ended 事件触发时调用 videoEnded 函数；对 video 元素监听 play 事件（打开页面时将自动播放视频，开始播放视频时将触发该事件，或者当用户单击本案例页面中的暂停按钮后再次单击播放按钮时也会触发该事件），指定 play 事件触发时调用 videoPlay 函数；对 video 元素监听 pause 事件（当用户单击暂停按钮时将触发该事件），指定 pause 事件

触发时将调用 `videoPause` 函数；对 `video` 元素监听 `timeupdate` 事件（在播放视频的过程中，当前播放位置改变时将触发该事件），指定 `timeupdate` 事件触发时将调用 `updateProgress` 函数；对 `video` 元素监听 `error` 事件（在播放视频的过程中，任何时刻，包括装载视频数据的时候，只要发生错误，就触发该事件），指定 `error` 事件触发时将调用 `catchError` 函数。指定当鼠标指针移动到模仿 `progress` 元素的 `div` 元素上时调用 `progress_mouseover` 函数，当鼠标指针从模仿 `progress` 元素的 `div` 元素上移出时调用 `progress_mouseout` 函数，当鼠标单击模仿 `progress` 元素的 `div` 元素时调用 `progress_click` 函数，当鼠标指针移动到 `progress` 元素上时调用 `progress_mouseover` 函数，当鼠标指针从 `progress` 元素上移出时调用 `progress_mouseout` 函数，当单击 `progress` 元素时调用 `playPercent_click` 函数。

在装载视频数据或播放视频的过程中，如果发生错误，则调用 `catchError` 函数，在该函数中将根据发生错误的错误状态而弹出对应的错误信息。

当浏览器获取完播放视频的时间长度和字节数时将触发 `loadedmetadata` 事件，该事件的触发表示浏览器能够正常播放 `video` 元素中的视频，这时将调用 `loadedmetadata` 函数。在该函数中指定播放按钮的按钮文字为“暂停”，设置播放按钮为有效状态，调用 `video` 元素的 `play` 方法开始播放视频，同时设置页面中的所有按钮为显示状态（通过 `buttonDiv.style.display="block"` 语句）。

单击播放按钮时将调用 `PlayOrPause` 函数，在该函数中首先判断视频文件处于暂停状态还是播放状态，如果视频处于暂停状态则播放视频，播放速率为变量 `speed` 的变量值（默认状态为 1），根据变量 `muted` 的变量值来指定视频是否为静音状态（默认状态下该变量值为 `false`，表示非静音），设定视频音量为变量 `volume` 的变量值（默认状态下该值为 1），如果视频处于播放状态则暂停视频。

当视频播放结束时将调用 `videoEnded` 函数，该函数将播放位置移动到视频开始播放的位置，同时暂停视频播放。

当页面打开，自动播放视频或单击播放按钮播放视频时将调用 `videoPlay` 函数，在该函数中设定播放按钮的文字为“暂停”，其他所有按钮均为有效状态。

单击暂停按钮时将调用 `videoPause` 函数，在该函数中设定播放按钮的按钮文字为“播放”，同时将页面中所有其他按钮均设为无效状态。

在播放视频的过程中，视频的播放位置发生改变时将调用 `updateProgress` 函数，在该函数中设置 `progress` 元素的元素值为视频当前已播放的时间除以视频总的播放时间，同时将模仿 `progress` 元素的 `div` 元素中的文字设定为“视频当前已播放的时间 / 视频总的播放时间”。在设定 `div` 元素文字时将调用一个 `calcTime` 函数，该函数的功能为将一个给定的时间段转换为“HH:MM:SS”的时间字符串（譬如 2 小时 18 分 18 秒为 02:18:18），然后返回该时间字符串。

当鼠标指针移动到 `progress` 元素或模仿 `progress` 元素的 `div` 元素上时将调用 `progress_mouseover` 函数，该函数将显示播放时间的 `div` 元素移动到鼠标指针处，同时显示播放时间字符串（例如 00:01:01/00:02:30，表示当前已播放 1 分 1 秒，视频总长度为 2 分 30 秒）。

当鼠标指针从 progress 元素或模仿 progress 元素的 div 元素上移出时将调用 progress\_mouseout 函数, 该函数隐藏显示播放时间的 div 元素。

单击 progress 元素时调用 playPercent\_click 函数将视频播放位置调整到单击鼠标位置, 视频从单击鼠标处开始播放。

单击模仿 progress 元素的 div 元素时调用 progress\_click 函数将视频播放位置调整到单击鼠标位置, 视频从单击鼠标处开始播放。

单击加速播放按钮时将调用 SpeedUp 函数, 该函数将视频的播放速率 +1, 然后将全局变量 speed 的值设定为修改后的播放速率。

单击减速播放按钮时将调用 SpeedDown 函数, 在该函数中将视频的播放速率 -1, 如果播放速率 -1 后变成负值时, 则把播放速率设为 0, 然后将全局变量 speed 的值设定为修改后的播放速率。

单击慢动作按钮时将调用 SlowPlay 函数, 在该函数中先根据按钮文字进行判断, 如果按钮文字为“慢动作”, 则将播放速率设定为 0.5 (正常播放速率的一半), 设定按钮文字为“正常速度”, 加速播放按钮与减速播放按钮均为无效状态; 如果按钮文字为“正常速度”, 则将播放速率设定为 1 (正常播放速率), 设定加速播放按钮与减速播放按钮均为有效状态, 全局变量 speed 的值为当前播放速率。

单击静音按钮时将调用 setMute 函数, 在该函数中首先判断视频的静音状态, 如果视频为非静音状态, 则设定视频为静音状态, 静音按钮文字为“取消静音”; 如果视频为静音状态, 则设定视频为非静音状态, 静音按钮文字为“静音”, 同时根据当前静音状态设置全局变量 muted 的值 (静音状态时该值为 true, 否则为 false)。

单击增大音量时调用 VolumeUp 函数将当前视频音量 +0.1, 同时将全局变量 volume 的值设定为修改后的视频音量值。

单击降低音量按钮时调用 VolumeDown 函数将当前视频音量 -0.1, 同时将全局变量 volume 的值设定为修改后的视频音量值。

## 5.2 案例 14: 对视频使用实时回放功能

### 5.2.1 案例概述

本案例在案例 13 的基础上进行修改, 添加一个回放按钮, 在播放视频的过程中, 在任何时刻单击回放按钮, 视频将从当前位置向视频开始位置方向进行回放, 一直回放到视频开始位置。

### 5.2.2 页面显示效果

先看一下本案例的页面显示效果, 本案例的页面以案例 13 的页面为基础, 添加一个回放按钮, 如图 5-7 所示。





图 5-7 添加了回放按钮的案例页面

### 5.2.3 代码剖析

接下来看一下为了实现本案例中的视频回放功能，对案例 13 中的代码进行了哪些必要的改变。

#### 1. HTML 页面代码

首先在案例 13 的 HTML 页面中添加一个回放按钮，代码如下所示。

```
<button id="btnPlayBack" onclick="PlayBack()" disabled/> 回放 </button>
```

修改后本案例的 HTML 页面代码如代码清单 5-3 所示。

代码清单 5-3 案例 14 的 HTML 页面代码

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8"/>
<title> 打造你自己的视频播放器 </title>
<style>
... 样式代码同案例 13 中的样式代码 ...
</style>
</head>
<body>
<video id="video" src="video.webm" width="800">
您的浏览器不支持 video 元素
</video>
<div id="buttonDiv">
<progress id="playPercent" max=100>
```

```

<div id="progress">
  <div id="progressValue"></div>
</div>
</progress>
<button id="btnPlay" onclick="PlayOrPause()" disabled/> 播放 </button>
<button id="btnSpeedUp" onclick="SpeedUp()" disabled/> 加速播放 </button>
<button id="btnSpeedDown" onclick="SpeedDown()" disabled/> 减速播放 </button>
<button id="btnSlowPlay" onclick="SlowPlay()" disabled/> 慢动作 </button>
<button id="btnMute" onclick="setMute()" disabled/> 静音 </button>
<button id="btnVolumeUp" onclick="VolumeUp()" disabled/> 增大音量 </button>
<button id="btnVolumeDown" onclick="VolumeDown()" disabled/> 降低音量
</button>
<button id="btnPlayBack" onclick="PlayBack()" disabled/> 回放 </button>
</div>
<div id="showTime">
</div>
</body>
<script type="text/javascript">
... 脚本代码稍后介绍 ...
</script>
</html>

```

## 2. JavaScript 脚本代码

接下来看一下在本案例的 JavaScript 脚本代码中，为了实现视频回放功能，对案例 13 中的 JavaScript 脚本代码做了哪些修改。

首先，在脚本代码开始处的全局变量定义中添加对全局变量 `direction` 的定义，并且将变量值初始化为 1。该全局变量表示视频的播放方向，值为 1 时表示视频向视频结尾处正向播放，值为 -1 时表示视频向视频开始处回放，代码如下所示。

```
var direction=1;
```

页面打开后，在自动播放视频或单击播放按钮播放视频时调用的 `videoPlay` 函数中设定回放按钮为有效状态，如果全局变量 `direction` 的值为 -1（代表将视频进行回放），则设定每 200 毫秒调用一次 `playBack1` 函数，在该函数中将视频当前播放位置向视频开始处回退一个单位，以实现视频回放的功能，修改后的 `videoPlay` 函数如下所示。

```

function videoPlay(ev)
{
  var btnPlay=document.getElementById("btnPlay");
  btnPlay.innerHTML=" 暂停 ";
  document.getElementById("btnSpeedUp").disabled="";
  document.getElementById("btnSpeedDown").disabled="";
  document.getElementById("btnSlowPlay").disabled="";
  document.getElementById("btnMute").disabled="";
  document.getElementById("btnVolumeUp").disabled="";
  document.getElementById("btnVolumeDown").disabled="";
  document.getElementById("btnPlayBack").disabled="";
}

```

```

        if(direction== -1)
            functionId=setInterval(playBack1,200);
    }

```

单击暂停按钮时调用的 videoPause 函数将回放按钮设为无效状态，同时停止调用 playBack1 函数，修改后的 videoPause 函数如下所示。

```

function videoPause(ev)
{
    var btnPlay=document.getElementById("btnPlay");
    btnPlay.innerHTML=" 播放 ";
    document.getElementById("btnSpeedUp").disabled="disabled";
    document.getElementById("btnSpeedDown").disabled="disabled";
    document.getElementById("btnSlowPlay").disabled="disabled";
    document.getElementById("btnMute").disabled="disabled";
    document.getElementById("btnVolumeUp").disabled="disabled";
    document.getElementById("btnVolumeDown").disabled="disabled";
    document.getElementById("btnPlayBack").disabled="disabled";
    clearInterval(functionId);
}

```

添加 PlayBack 函数，代码如下所示。

```

function PlayBack(){
    var playBackBtn=document.getElementById("btnPlayBack");
    if(playBackBtn.innerHTML==" 回放 ")
    {
        functionId=setInterval(playBack1,200);
        playBackBtn.innerHTML=" 取消回放 ";
        direction=-1;
    }
    else
    {
        clearInterval(functionId);
        playBackBtn.innerHTML=" 回放 ";
        direction=1;
    }
}

```

在该函数中，首先获取页面上的回放按钮，如果回放按钮的按钮文字为“回放”，则设定每 200 毫秒调用一次 playBack1 函数，在该函数中将视频从当前播放位置向视频开始处回退一个单位，以实现视频回放功能，同时将回放按钮的按钮文字修改为“取消回放”，将全局变量 direction 的值设定为 -1（表示将视频向开始处回放）。如果回放按钮的按钮文字为“取消回放”，则停止调用 playBack1 函数，将回放按钮的按钮文字修改为“回放”，同时将全局变量 direction 的值设定为 1（表示将视频向结尾处正向播放）。

添加 playBack1 函数，代码如下所示。

```

function playBack1()
{

```

```

var playBackBtn=document.getElementById("btnPlayBack");
if(video.currentTime==0)
{
    playBackBtn.innerHTML=" 回放 ";
    clearInterval(functionId);
}
else
    video.currentTime-=1;
}

```

在回放过程中, playBack1 函数每隔 200 毫秒被调用一次, 在该函数中, 首先判定视频的当前播放位置是否处于视频开始处, 如果是的则停止调用 playBack1 函数, 否则将视频的当前播放位置回退 1 个单位。

## 5.3 案例 15: 对视频使用截图功能

### 5.3.1 案例概述

本案例在案例 13 与案例 14 的基础上继续扩展, 添加了视频截图功能, 当用户单击截图按钮时, 案例程序将把视频中播放的当前帧中的画面输出到页面的 canvas 元素中, 用户可以自行把 canvas 元素中的画面保存到磁盘文件中。

### 5.3.2 页面显示效果

首先来看一下本案例页面在浏览器中的显示效果, 本案例页面在案例 13 与案例 14 的基础上添加了一个截图按钮, 如图 5-8 所示。



图 5-8 添加了截图按钮的案例页面

单击截图按钮时, 案例程序将把视频当前帧中的画面输出到页面上的 canvas 元素中, 如图 5-9 所示。



图 5-9 单击截图按钮时案例程序将把视频当前帧中的画面输出到 canvas 元素中

### 5.3.3 案例知识点

视频截图的关键在于调用 canvas 元素的图形上下文对象的 drawImage 方法, 并将页面中的 video 元素指定为函数的第一个参数, 这样调用 drawImage 方法时, 该方法会自动将 video 元素中视频当前播放帧中的画面绘制在页面的 canvas 元素中, 代码如下所示。

```
ctx.drawImage(video, 0, 0, canvas.width, canvas.height);
```

### 5.3.4 代码剖析

接下来看一下为了实现本案例中的视频回放功能, 对案例 14 中的代码进行了哪些必要的改变。

#### 1. HTML 页面代码与样式代码

首先, 向案例 14 的 HTML 页面中添加一个截图按钮及一个 canvas 元素, 代码如下所示。

```
<button id="btnCatchPicture" onclick="CatchPicture()" disabled />
截图 </button>
... 中间代码为案例 13 中原有代码, 故此略去 ...
<canvas id="canvas" width="800" height="256">
</canvas>
```

同时,在样式代码中指定页面打开时 canvas 元素为隐藏状态,只有当用户单击截图按钮时该元素才变为可见状态,本案例中添加的样式代码如下所示。

```
canvas{
    display:none;
}
```

修改后案例中的 HTML 页面代码与样式代码如代码清单 5-4 所示。

代码清单 5-4 案例 15 的 HTML 页面代码与样式代码

---

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8"/>
<title> 打造你自己的视频播放器 </title>
<style>
div#buttonDiv{
    width:800px;
    display:none;
}
progress#playPercent{
    width:800px;
}
div#progress{
    width:800px;
    height:20px;
    background:#ccc;
}
div#progressValue{
    width:0%;
    height:20px;
    background:green;
    cursor:default;
}
div#showTime{
    z-index:2;
    display:none;
    position:absolute;
    font-size:10px;
}
canvas{
    display:none;
}
</style>
</head>
<body>
<video id="video" src="video.webm" width="800">
您的浏览器不支持 video 元素
</video>
<div id="buttonDiv">
<progress id="playPercent" max=100>
```

```

        <div id="progress">
            <div id="progressValue"></div>
        </div>
    </progress>
    <button id="btnPlay" onclick="PlayOrPause()" disabled/> 播放 </button>
    <button id="btnSpeedUp" onclick="SpeedUp()" disabled/> 加速播放 </button>
    <button id="btnSpeedDown" onclick="SpeedDown()" disabled/> 减速播放 </button>
    <button id="btnSlowPlay" onclick="SlowPlay()" disabled/> 慢动作 </button>
    <button id="btnMute" onclick="setMute()" disabled/> 静音 </button>
    <button id="btnVolumeUp" onclick="VolumeUp()" disabled/> 增大音量 </button>
    <button id="btnVolumeDown" onclick="VolumeDown()" disabled/> 降低音量
</button>
<button id="btnPlayBack" onclick="PlayBack()" disabled/> 回放 </button>
<button id="btnCatchPicture" onclick="CatchPicture()" disabled /> 截图
</button>
</div>
<div id="showTime">
</div>
<canvas id="canvas" width="800" height="256">
</canvas>
</body>
<script type="text/javascript">
... 脚本代码稍后介绍 ...
</script>
</html>

```

---

## 2. JavaScript 脚本代码

接下来看一下在本案例的 JavaScript 脚本代码中，为了实现视频截图功能，对案例 14 中的 JavaScript 脚本代码做出了哪些修改。

页面打开后，自动播放视频或单击播放按钮播放视频时调用的 videoPlay 函数中的截图按钮为有效状态，修改后的 videoPlay 函数如下所示。

```

function videoPlay(ev)
{
    var btnPlay=document.getElementById("btnPlay");
    btnPlay.innerHTML=" 暂停 ";
    document.getElementById("btnSpeedUp").disabled="";
    document.getElementById("btnSpeedDown").disabled="";
    document.getElementById("btnSlowPlay").disabled="";
    document.getElementById("btnMute").disabled="";
    document.getElementById("btnVolumeUp").disabled="";
    document.getElementById("btnVolumeDown").disabled="";
    document.getElementById("btnPlayBack").disabled="";
    document.getElementById("btnCatchPicture").disabled="";
    if(direction== -1)
        functionId=setInterval(playBack1,200);
}

```

在单击暂停按钮时调用的 videoPause 函数中将截图按钮设为无效状态，修改后的

videoPause 函数如下所示。

```
function videoPause(ev)
{
    var btnPlay=document.getElementById("btnPlay");
    btnPlay.innerHTML="播放";
    document.getElementById("btnSpeedUp").disabled="disabled";
    document.getElementById("btnSpeedDown").disabled="disabled";
    document.getElementById("btnSlowPlay").disabled="disabled";
    document.getElementById("btnMute").disabled="disabled";
    document.getElementById("btnVolumeUp").disabled="disabled";
    document.getElementById("btnVolumeDown").disabled="disabled";
    document.getElementById("btnPlayBack").disabled="disabled";
    document.getElementById("btnCatchPicture").disabled="disabled";
    clearInterval(functionId);
}
```

添加 CatchPicture 函数，当用户单击截图按钮时案例程序将调用该函数，将视频当前播放帧中的画面输出到 video 元素中，同时设定 canvas 元素为可见状态，代码如下所示。

```
function CatchPicture()
{
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext('2d');
    canvas.width=video.videoWidth;
    canvas.height=video.videoHeight;
    ctx.drawImage(video,0,0,canvas.width,canvas.height);
    canvas.style.display="block";
}
```

## 5.4 案例 16：打造自己的网页音频播放器

### 5.4.1 案例概述

本节通过一个制作网页音频播放器的案例向读者介绍如何使用 HTML 5 中的 audio 元素来播放音频文件，如何利用 audio 元素的各种属性、事件与方法来定制属于自己的网页媒体播放器。

### 5.4.2 页面显示效果

首先来看一下本案例在浏览器中的页面显示效果。本案例通过页面中放置的一个 audio 元素来播放视频，不指定该 audio 元素的 controls 属性以隐藏浏览器自己提供的播放控件，在视频下部使用 HTML5 中的 progress 元素作为播放进度条，在进度条下部显示几个按钮，按钮文字分别为“播放”、“后退”、“前进”、“重新开始”、“静音”、“增大音量”和“降低音量”。



在一个不支持 audio 元素的浏览器中，页面的显示效果如图 5-10 所示。



图 5-10 案例页面在不支持 audio 元素的浏览器中的显示效果

在支持 audio 元素的浏览器（本案例中使用 Google Chrome 10 浏览器）中，页面的显示效果如图 5-11 所示。页面上部为一个 progress 元素，用来表示播放进度，页面下部为几个用来对播放音频进行控制的按钮。

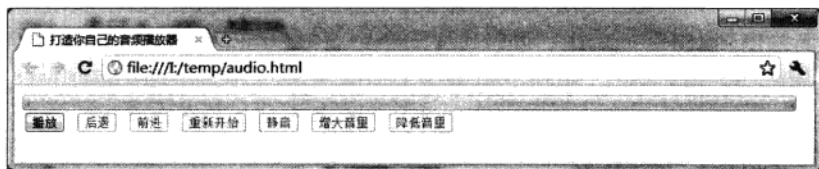


图 5-11 在 Google Chrome 10 浏览器中打开页面时的显示效果

单击播放按钮后，开始播放页面中指定的音频文件，页面右下部显示音频文件的总播放时长与当前播放时长，如图 5-12 所示。

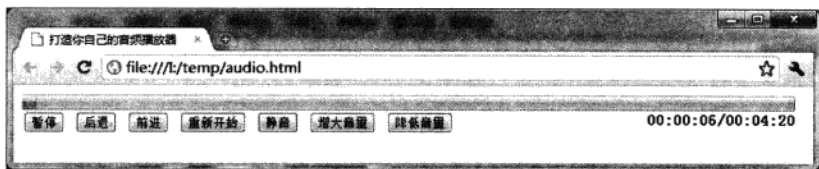


图 5-12 单击播放按钮后开始播放页面中指定的音频文件

### 5.4.3 案例知识点

在详细分析页面代码之前，先介绍一下本案例中所使用的 audio 元素的属性、方法与事件，audio 元素的完整说明请参阅笔者所著的《HTML 5 与 CSS 3 权威指南》。

□ Src 属性。

在该属性中指定播放音频的 URL 地址。

□ played 属性、paused 属性和 ended 属性。

可以通过 audio 元素的 played 属性返回一个 TimeRanges 对象，从该对象中可以读取音频的已播放部分的时间段。开始时间为已播放部分的开始时间，结束时间为已播放部分的结束时间。

可以通过 audio 元素的 paused 属性返回一个布尔值，表示音频是否处于暂停播放中，true 表示音频暂停播放，false 表示音频正在播放。

可以通过 audio 元素的 end 属性来返回一个布尔值，表示是否播放完毕，true 表示音频播放完毕，false 表示还没有播放完毕。

这三个属性均为只读属性。

❑ volume 属性与 muted 属性。

可以通过 video 元素的 volume 属性读取或修改视频的播放音量，范围为 0 到 1，0 为静音，1 为最大音量。

可以通过 video 元素的 muted 属性读取或修改视频的静音状态，该值为布尔值，true 表示处于静音状态，false 表示处于非静音状态。

❑ play 方法。

使用 play 方法来播放音频，自动将元素的 paused 属性的属性值变为 false。

❑ pause 方法。

使用 pause 方法来暂停播放，自动将元素的 paused 属性的属性值变为 true。

❑ ended 事件。

当音频文件播放结束或停止时将触发该事件。

❑ play 事件。

当使用 audio 元素的 play 方法播放音频时将触发该事件，或者音频数据被下载完毕且 audio 元素被设为 autoplay（自动播放）属性后自动播放音频时将触发该事件。

❑ pause 事件。

当执行了 audio 元素的 pause 方法暂停音频的播放时将触发该事件。

❑ timeupdate 事件。

当前播放位置被改变时将触发该事件。播放位置的改变可能是音频播放过程中的自然改变，也可能是人为改变，或者是由于播放不能连续而发生的跳变。

#### 5.4.4 代码剖析

接下来具体分析本案例的整个实现过程及实现代码。

##### 1. HTML 页面代码与样式代码

本案例的 HTML 页面代码部分比较简单，首先使用一个 audio 元素来播放音频，在不支持 audio 元素的浏览器中将直接显示文字“您的浏览器不支持 audio 元素”。在浏览器的下部使用一个 progress 元素作为播放进度条。在 progress 元素的下部放置一些按钮，文字分别为“播放”、“后退”、“前进”、“重新开始”、“静音”、“增大音量”和“降低音量”。在页面的右下部还使用了一个 div 元素，该 div 元素的 id 为“showTime”，在该 div 元素中显示音频总的播放时间与当前已播放部分的播放时间。

本案例的 HTML 页面代码与样式代码如代码清单 5-5 所示。

代码清单 5-5 本案例的 HTML 页面代码与样式代码

---

```

<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8"/>
<title> 打造你自己的音频播放器 </title>
<style>
div#buttonDiv{
    width:800px;
}
progress#playPercent{
    width:800px;
}
div#divLeft{
    float:left;
}
div#divRight{
    float:right;
    font-weight:bold;
}
</style>
<script type="text/javascript">
... 脚本代码稍后介绍 ...
</script>
</head>
<body onload="window_onload()">
<audio id="audio" src=" 昔日舞曲 .mp3">
您的浏览器不支持 audio 元素
</audio>
<div id="buttonDiv">
<div id="divLeft"><progress id="playPercent" max=100></progress></div>
<div id="divRight"><span id="showTime"></span></div>
<button id="btnPlay" onclick="btnPlay_click()" disabled/> 播放 </button>
<button id="btnBackPlay" onclick="btnBackPlay_click()" disabled/> 后退 </button>
<button id="btnForwardPlay" onclick="btnForwardPlay_click()" disabled/> 前进
</button>
<button id="btnRestart" onclick="btnRestart_click()" disabled/> 重新开始 </button>
<button id="btnMute" onclick="setMute()" disabled/> 静音 </button>
<button id="btnVolumeUp" onclick="VolumeUp()" disabled/> 增大音量 </button>
<button id="btnVolumeDown" onclick="VolumeDown()" disabled/>
降低音量 </button>
</div>
</body>
</html>

```

---

## 2. JavaScript 脚本代码

接下来的代码清单 5-6 为本案例的完整的 JavaScript 脚本代码部分，稍后将对这个 JavaScript 脚本代码进行详细介绍。

代码清单 5-6 本案例的完整的 JavaScript 脚本代码

```

<script type="text/javascript">
var audio;
var showTime;
function window_onload()
{
    if (window.HTMLAudioElement)
    {
        document.getElementById("btnPlay").disabled = "";
        audio = document.getElementById("audio");
        progress=document.getElementById("playPercent");
        progress.onclick=progress_click;
    }
    else
    {
        document.getElementById("buttonDiv").style.display="none";
    }
}
function btnPlay_onclick()
{
    audio = document.getElementById("audio");
    if(audio.paused)
        audio.play();
    else
        audio.pause();
    audio.addEventListener('ended',audioEnded,false);
    audio.addEventListener('play',audioPlay,false);
    audio.addEventListener('pause',audioPause,false);
    audio.addEventListener('timeupdate',updateProgress,false);
}
function audioEnded(ev)
{
    audio.currentTime=0;
    this.pause();
}
function audioPlay(ev)
{
    var btnPlay=document.getElementById("btnPlay");
    btnPlay.innerHTML=" 暂停 ";
    document.getElementById("btnBackPlay").disabled="";
    document.getElementById("btnForwardPlay").disabled="";
    document.getElementById("btnRestart").disabled="";
    document.getElementById("btnMute").disabled="";
    document.getElementById("btnVolumeUp").disabled="";
    document.getElementById("btnVolumeDown").disabled="";
}
function audioPause(ev)
{
    var btnPlay=document.getElementById("btnPlay");

```

```

    btnPlay.innerHTML=" 播放 ";
    document.getElementById("btnBackPlay").disabled="disabled";
    document.getElementById("btnForwardPlay").disabled="disabled";
    document.getElementById("btnRestart").disabled="disabled";
    document.getElementById("btnMute").disabled="disabled";
    document.getElementById("btnVolumeUp").disabled="disabled";
    document.getElementById("btnVolumeDown").disabled="disabled";
}
function updateProgress()
{
    var value=Math.round((Math.floor(audio.currentTime)/
    Math.floor(audio.duration))*100,0);
    var progress = document.getElementById('playPercent');
    progress.value = value;
    var showTime=document.getElementById("showTime");
    showTime.innerHTML=calcTime(Math.floor(audio.currentTime))+'/' +
    calcTime(Math.floor(audio.duration));
}
function calcTime(time)
{
    var hour;
    var minute;
    var second;
    hour=String(parseInt(time/3600,10));
    if (hour.length == 1)    hour    = '0' + hour;
    minute=String(parseInt((time%3600)/60,10));
    if (minute.length == 1)    minute    = '0' + minute;
    second=String(parseInt(time%60,10));
    if (second.length == 1)    second    = '0' + second;
    return hour+":"+minute+":"+second;
}
function progress_click(evt)
{
    playPercent=document.getElementById("playPercent");
    audio.currentTime = audio.duration * (evt.offsetX / playPercent.clientWidth);
}
function btnBackPlay_click()
{
    audio.currentTime -= 30.0;
}
function btnForwardPlay_click()
{
    audio.currentTime += 30.0;
}
function btnRestart_click()
{
    audio.currentTime=0;
}
function setMute()
{

```

```

    if(!audio.muted)
    {
        audio.muted=true;
        document.getElementById("btnMute").innerHTML="取消静音";
    }
    else
    {
        audio.muted=false;
        document.getElementById("btnMute").innerHTML="静音";
    }
}
function VolumeUp()
{
    if(audio.volume<1)
        audio.volume+=0.1;
}
function VolumeDown()
{
    if(audio.volume>0)
        audio.volume-=0.1;
}
</script>

```

在这段脚本代码的开始处，定义了本案例中使用的两个全局变量，其中全局变量 `audio` 代表页面中的 `audio` 元素，全局变量 `showTime` 代表页面中用来显示音频总的播放时间与当前已播放时间的 `div` 元素。

接下来，指定在打开页面时直接执行的一些 JavaScript 脚本代码。

在打开页面时，调用 `window_onload` 函数执行页面元素的初始化工作。

在该函数中，首先使用 `window` 对象的 `HTMLAudioElement` 属性来判断浏览器是否支持 `audio` 元素，如果不支持就只在浏览器中显示一段“您的浏览器不支持 `audio` 元素”文字，不显示页面中的其他任何元素。

如果浏览器支持 `audio` 元素，就将页面中的播放按钮设定为有效状态，同时设定单击 `progress` 元素（播放进度条）时调用 `progress_click` 函数。

单击播放按钮时将调用 `btnPlay_onclick` 函数，在该函数中首先判断音频文件处于暂停状态还是播放状态，如果音频处于暂停状态则播放音频，如果音频处于播放状态则暂停音频。然后对 `audio` 元素监听 `ended` 事件（当音频播放完毕或者停止播放时将触发该事件），指定 `ended` 事件触发时调用 `audioEnded` 函数；对 `audio` 元素监听 `play` 事件（打开页面时将自动播放音频，开始播放音频时将触发该事件，或者当用户单击本案例页面中的暂停按钮后再次单击播放按钮时也会触发该事件），指定 `play` 事件触发时调用 `audioPlay` 函数；对 `audio` 元素监听 `pause` 事件（当用户单击暂停按钮时将触发该事件），指定 `pause` 事件触发时将调用 `audioPause` 函数；对 `audio` 元素监听 `timeupdate` 事件（在播放音频的过程中，当前播放位置改变时将触发该事件），指定 `timeupdate` 事件触发时将调用 `updateProgress` 函数。

当音频播放结束时将调用 `audioEnded` 函数，该函数将播放位置移动到音频开始播放的位置，同时暂停音频播放。

打开页面后，自动播放音频或单击播放按钮播放音频时将调用 `audioPlay` 函数，在该函数中设定播放按钮的文字为“暂停”，其他所有按钮均为有效状态。

单击暂停按钮时将调用 `audioPause` 函数，在该函数中设定播放按钮的按钮文字为“播放”，同时将页面中所有其他按钮均设为无效状态。

在播放音频的过程中，音频的播放位置发生改变时将调用 `updateProgress` 函数，在该函数中设置 `progress` 元素的元素值为音频当前已播放的时间除以音频总的播放时间，同时 will 将页面右下部的 `div` 元素中的文字设定为“音频当前已播放的时间 / 音频总的播放时间”。在设定 `div` 元素文字时将调用一个 `calcTime` 函数，该函数的功能为将一个给定的时间段转换为“HH:MM:SS”的时间字符串（譬如 2 小时 18 分 18 秒为 02:18:18），然后返回该时间字符串。

单击 `progress` 元素（用来显示页面上的音频播放进度条）时调用 `progress_click` 函数将音频播放位置调整到单击鼠标的位置，音频从单击鼠标处开始播放。

单击后退按钮时调用 `btnBackPlay_click` 函数将音频当前播放位置向后（音频开始处方向）调整 30 秒，音频从调整后的播放位置处开始播放。

单击前进按钮时调用 `btnForwardPlay_click` 函数将音频当前播放位置向前（音频结束处方向）调整 30 秒，音频从调整后的播放位置处开始播放。

单击重新开始按钮时调用 `btnRestart_click` 函数将音频当前播放位置调整到音频开始处，音频从头开始播放。

单击静音按钮时将调用 `setMute` 函数，在该函数中首先判断音频的静音状态，如果音频为非静音状态，则设定音频为静音状态，静音按钮文字为“取消静音”；如果音频为静音状态，则设定音频为非静音状态，静音按钮文字为“静音”，同时根据当前静音状态设置全局变量 `muted` 的值（静音状态时该值为 `true`，否则为 `false`）。

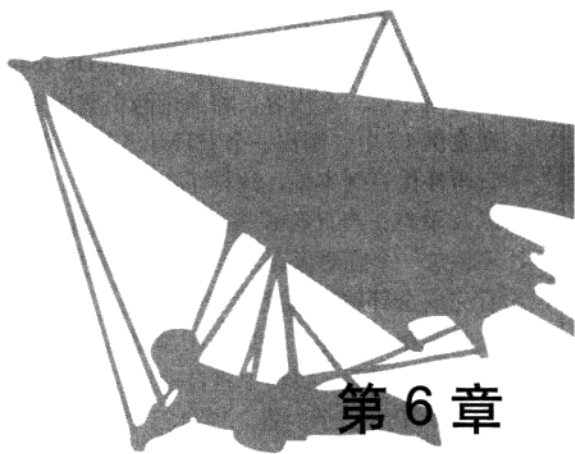
单击增大音量时调用 `VolumeUp` 函数将当前音频音量 +0.1。

单击降低音量按钮时调用 `VolumeDown` 函数将当前视频音量 -0.1。

## 5.5 本章小结

本章通过 4 个案例来详细阐述了 HTML 5 中 `video` 元素与 `audio` 元素的基本知识及使用方法，力求使读者通过本章内容对 HTML 5 中的 `video` 元素与 `audio` 元素有了一个比较全面的认识，进而制作出属于自己的网页视频播放器与音频播放器。

下一章将通过几个案例来详细阐述 HTML 5 中的一个非常重要的内容——本地存储。这部分内容有三方面内容：Web Storage、Web Database 和离线应用程序。针对每方面内容，通过案例来具体介绍，力求使读者全面了解 HTML 5 中的本地存储，能够将其运用在实际开发过程之中。



## 第 6 章

# 本地存储

### 本章内容

- ☐ 案例 17: 制作 HTML 5 版本的日程提醒簿
- ☐ 案例 18: 临时保存页面中的输入内容
- ☐ 案例 19: 使用 HTML 5 制作 Web 应用程序的演示版
- ☐ 案例 20: 使用客户端 session
- ☐ 案例 21: 将本地数据库中的数据提交到服务器端
- ☐ 案例 22: 制作可以离线使用的日程提醒簿
- ☐ 本章小结



本章介绍 HTML 5 中与本地存储（将用户数据或网页内容存储在客户端计算机中）相关的三个内容——Web Storage、Web Database（本地数据库）与离线 Web 应用程序的开发。

对于第一个内容，将通过两个案例来重点介绍如何使用 HTML 5 中的 Web Storage 功能。在案例 17 中，制作一个 HTML 5 版本的日程提醒簿，利用 Web Storage 功能将用户输入的每日事件保存到本地，这样下次用户打开浏览器访问本页面时，日程提醒簿中会显示当日日期以及用户在当日有哪些必须要处理的事件。在案例 18 中，将向读者展示如何向 Web 应用程序添加一个临时保存功能，这样当用户使用 Web 应用程序的时候，如果在某个输入页面中没有完成全部输入功能，需要中断一下输入过程（譬如突然有事外出），可以将当前输入内容临时保存在本地，然后关闭浏览器。当用户要继续输入时，打开浏览器并访问之前保存了输入内容的页面，页面中将自动显示用户保存的输入内容。

对于第二个内容，通过三个案例来向读者展示 HTML 5 中 Web Database 的功能及其使用方法。在案例 19 中，从制作一个演示版的 Web 应用程序的角度出发，向读者展示如何将用户在表单中输入的数据保存到客户端本地数据库中，以及如何对客户端本地数据库使用增删查改功能。在第 20 案例中，将在个 Web 应用程序的演示版中使用 session 功能，以此向读者介绍如何在 HTML 页面之间通过使用 session 功能在多个页面之间传递数据，而不需要使用服务器端的 session，使用 HTML 5 中的 sessionStorage 即可实现客户端的 session 功能。在第 21 案例中，将向读者介绍如何将客户端本地数据库中的数据提交到服务器中。

对于第三个内容，对在案例 17 中制作的日程提醒簿添加离线应用程序功能，这样用户第一次访问案例页面之后，即使不上网也能使用该案例中所制作的日程提醒簿应用程序。

## 6.1 案例 17：制作 HTML 5 版本的日程提醒簿

### 6.1.1 案例概述

本案例将制作一个 HTML 5 版本的日程提醒簿。用户打开浏览器时，案例程序将在一个日程提醒簿中显示本日日期与用户在本日有哪些必须要处理的事件，用户可以在页面的选择日期文本框中输入其他日期，然后在日程提醒簿中输入选择的日期所要处理的事件并保存，这样当用户在所选择的日期打开浏览器时，浏览器将在日程提醒簿中显示在该日期用户所需要处理的事件。

### 6.1.2 页面显示效果

首先来看一下案例页面在浏览器中的显示效果。该页面中具有一个选择日期文本框与一个日程提醒簿，页面打开时选择日期文本框与日程提醒簿中将显示本日日期，如图 6-1 所示（使用的浏览器为 Google Chrome 浏览器）。

用户可以在选择日期文本框中输入其他日期，输入其他日期后日程提醒簿中将显示所选择的日期，如图 6-2 所示。

用户可以在该日程提醒簿中输入当前要处理的事件，并单击保存按钮进行保存，如图 6-3 所示。

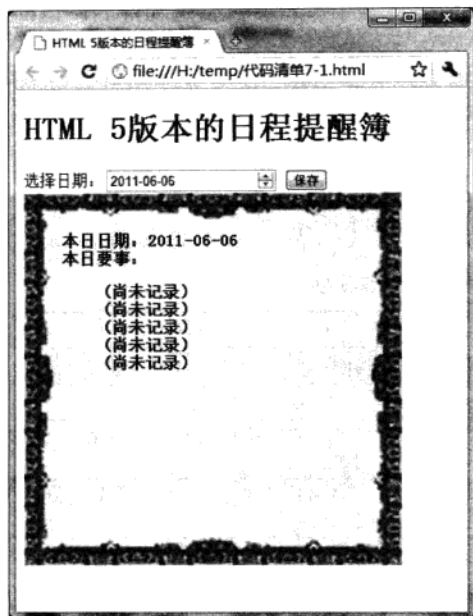


图 6-1 案例页面打开时的显示效果

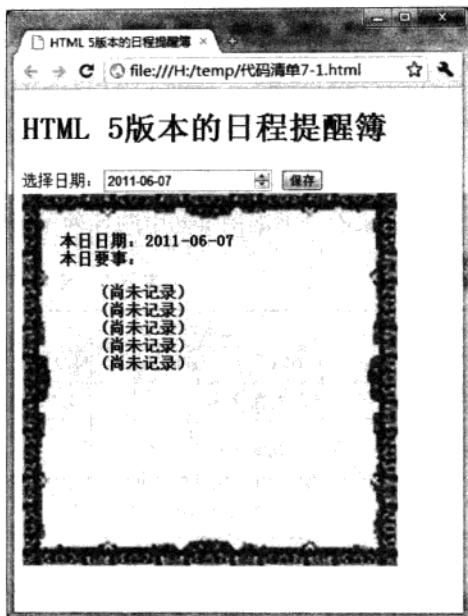


图 6-2 HTML 5 版本的日程提醒簿

这样当用户在所选择的日期内打开浏览器并访问该页面时，该页面的日程提醒簿中将显示用户所保存的事件，页面显示效果同图 6-3。

### 6.1.3 案例知识点

接下来看一下本案例中使用到的 localStorage 的一些属性与方法，关于 localStorage 的详细介绍请参阅笔者所著的《HTML 5 与 CSS 3 权威指南》一书。

#### ○ localStorage.setItem (key,value)

使用 localStorage 对象的 setItem 方法将数据永久保存在客户端计算机中，保存数据时按“键名/键值”的形式进行。将第一个参数指定为键名，将第二个参数指定为键值。

保存时不允许重复保存相同的键名。保存后可以修改键值，但不允许修改键名（只能重新取

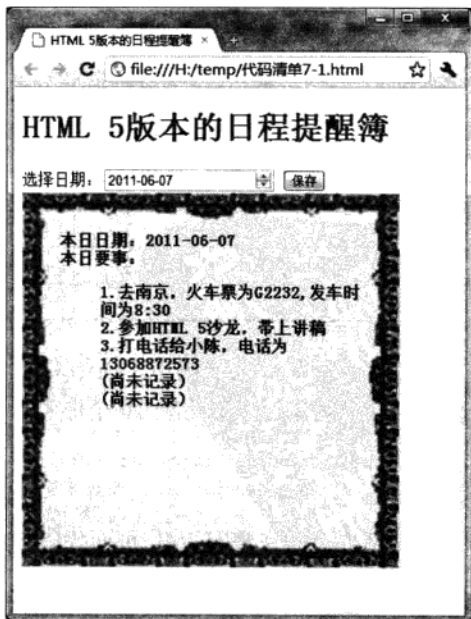


图 6-3 在选择日期中输入当日事件并保存

键名，然后再保存键值)。

○ 变量 = localStorage.getItem(key)

使用 localStorage 对象的 getItem 方法将数据读取到变量中，将参数指定为键名，返回键值并保存到变量中。

### 6.1.4 代码剖析

#### 1. HTML 页面代码与样式代码

首先来看一下案例页面中的各种元素及其说明，如表 6-1 所示。

表 6-1 案例页面中的各种元素及其说明

元素	元素种类	显示文字	说明
页面标题	h1	HTML 5 版本的日程提醒簿	
选择日期文字	页面文字	选择日期:	
选择日期文本框	input type="date"		用户在文本框内输入需要保存事件的日期
保存按钮	input type="button"	保存	单击该按钮时将用户在日程提醒簿中输入的事件保存
日程提醒簿	div		以下元素均显示在日程提醒簿 div 内
本日日期文字	页面文字	本日日期:	
本日日期	span		在本日日期 span 内显示打开浏览器并访问本页面的当日日期
本日要事	页面文字	本日要事:	
事件列表	ul		以下 5 个列表项目元素均为事件列表 ul 元素的子元素，事件列表 ul 元素的 contentEditable 属性值为 true (允许用户修改列表项目 li 元素中的文字)
事件列表项目 1	li	(尚未记录)	用户可以修改元素内文字
事件列表项目 2	li	(尚未记录)	用户可以修改元素内文字
事件列表项目 3	li	(尚未记录)	用户可以修改元素内文字
事件列表项目 4	li	(尚未记录)	用户可以修改元素内文字
事件列表项目 5	li	(尚未记录)	用户可以修改元素内文字

在本页面的样式代码中，使用 CSS 3 中的图像边框以一个图像文件作为 div 元素的边框图像，同时指定边框宽度为 10 个像素，代码如下所示。

```
-webkit-border-image: url(日历背景.png) 10;  
-moz-border-image: url(日历背景.png) 10;
```

案例页面的完整 HTML 页面代码与样式代码如代码清单 6-1 所示。

代码清单 6-1 案例页面完整的 HTML 页面代码与样式代码

---

```

<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title>HTML 5 版本的日程提醒簿 </title>
<style>
div{
    -webkit-border-image: url( 日历背景.png) 10;
    -moz-border-image: url( 日历背景.png) 10;
    width:300px;
    height:300px;
    padding:35px;
    background:#eee;
    font-weight:bold;
}
li{
    list-style:none;
}
</style>
<script type="text/javascript">
... 脚本代码稍后介绍 ...
</script>
</head>
<body onload="window.onload()">
<h1>HTML 5 版本的日程提醒簿 </h1>
选择日期: <input id="date1" type="date" onchange="date_onchange()">
<input type="button" value=" 保存 " onclick="save()" /><br/>
<div>
本日日期: <span id="today"></span><br/>
本日要事: <br/>
<ul contentEditable="true">
<li id="li1">( 尚未记录 ) </li>
<li id="li2">( 尚未记录 ) </li>
<li id="li3">( 尚未记录 ) </li>
<li id="li4">( 尚未记录 ) </li>
<li id="li5">( 尚未记录 ) </li>
</ul>
</div>
</body>
</html>

```

---

## 2. JavaScript 脚本代码

接下来通过代码清单 6-2 来看一下本案例完整的 JavaScript 脚本代码, 稍后对这份脚本代码进行详细分析。

代码清单 6-2 本案例完整的 JavaScript 脚本代码

```

<script type="text/javascript">
var dateElement;
var today;
function window_onload()
{
    dateElement=document.getElementById("date1");
    today=document.getElementById("today");
    setToday();
}
function date_onchange()
{
    var obj;
    if(isNaN(Date.parse(dateElement.value)))
    {
        setToday();
        return;
    }
    today.innerHTML=dateElement.value;
    obj=JSON.parse(localStorage.getItem(dateElement.value));
    setInnerHTML(obj);
}
function save()
{
    var obj=new Object();
    obj.record=new Array();
    if(document.getElementById("li1").innerHTML!="( 尚未记录)")
        obj.record.push(document.getElementById("li1").innerHTML);
    if(document.getElementById("li2").innerHTML!="( 尚未记录)")
        obj.record.push(document.getElementById("li2").innerHTML);
    if(document.getElementById("li3").innerHTML!="( 尚未记录)")
        obj.record.push(document.getElementById("li3").innerHTML);
    if(document.getElementById("li4").innerHTML!="( 尚未记录)")
        obj.record.push(document.getElementById("li4").innerHTML);
    if(document.getElementById("li5").innerHTML!="( 尚未记录)")
        obj.record.push(document.getElementById("li5").innerHTML);
    localStorage.setItem(dateElement.value,JSON.stringify(obj));
}
function setInnerHTML(obj)
{
    if(obj==null||obj.record==null)
    {
        document.getElementById("li1").innerHTML="( 尚未记录)";
        document.getElementById("li2").innerHTML="( 尚未记录)";
        document.getElementById("li3").innerHTML="( 尚未记录)";
        document.getElementById("li4").innerHTML="( 尚未记录)";
        document.getElementById("li5").innerHTML="( 尚未记录)";
    }
    else

```

```

{
    if(obj.record[0]!=null)
        document.getElementById("li1").innerHTML=obj.record[0];
    else
        document.getElementById("li1").innerHTML="( 尚未记录 )";
    if(obj.record[1]!=null)
        document.getElementById("li2").innerHTML=obj.record[1];
    else
        document.getElementById("li2").innerHTML="( 尚未记录 )";
    if(obj.record[2]!=null)
        document.getElementById("li3").innerHTML=obj.record[2];
    else
        document.getElementById("li3").innerHTML="( 尚未记录 )";
    if(obj.record[3]!=null)
        document.getElementById("li4").innerHTML=obj.record[3];
    else
        document.getElementById("li4").innerHTML="( 尚未记录 )";
    if(obj.record[4]!=null)
        document.getElementById("li5").innerHTML=obj.record[5];
    else
        document.getElementById("li5").innerHTML="( 尚未记录 )";
}
}
function setToday()
{
    var date=new Date();
    var yearStr=String(date.getFullYear());
    var monthStr=String(date.getMonth()+1);
    var dateStr=String(date.getDate());
    if (monthStr.length == 1)    monthStr = '0' + monthStr;
    if (dateStr.length == 1) dateStr = '0' + dateStr;
    var str=yearStr+"-"+monthStr+"-"+dateStr;
    dateElement.value=str;
    today.innerHTML=dateElement.value;
    var obj=JSON.parse(localStorage.getItem(dateElement.value));
    setInnerHTML(obj);
}
</script>

```

在脚本代码的开始处定义了脚本代码中所使用的两个全局变量，其中变量 `dateElement` 表示页面中的选择日期文本框，变量 `today` 表示页面中用来显示本日日期的 `span` 元素。

在脚本代码中还调用以下几个函数。

#### ❑ `window_onload` 函数

页面打开时调用 `window_onload` 函数，在该函数中首先获取页面中的选择日期文本框并赋值给全局变量 `dateElement`，获取页面中用来显示本日日期的 `span` 元素并赋值给全局变量 `today`，然后调用 `setToday` 函数。

#### ❑ `setToday` 函数

在该函数中首先获取本日日期，并将本日日期以“YYYY-MM-DD”的形式显示在选择日期文本框与显示本日日期的 span 元素中，然后读取 localStorage 中保存的本日日期的事件，最后调用 setInnerHTML 函数。

#### □ setInnerHTML 函数

本函数接受一个 obj 参数，该参数为一个 JSON 对象，此对象具有一个 record 属性，属性值为一个数组，数组的大小为 5，数组中前面几项为字符串，后面几项为 NULL（可能全部为字符串，也可能全部为 NULL，但是字符串数组项始终位于 NULL 数组的前面几项，根据用户在 localStorage 中保存的该日日期的事件条数决定字符串在数组中个数），在每一个字符串中保存了用户在 localStorage 中保存的该日日期的事件内容。

在该函数中，首先判定用户是否在 localStorage 中存入了本日日期或所选日期的事件内容，如果 localStorage 中没有本日日期或所选日期的事件内容，将事件列表中每一个列表项中的文字都设定为“（尚未记录）”，否则按照顺序读入 obj 对象的 record 属性值数组中的每一个数组项中的内容，如果该数组项中内容不为 NULL，则将该数组项中的内容设置到事件列表中与之对应的列表项中，否则将与之对应的列表项文字设定为“（尚未记录）”。

#### □ date\_onchange 函数

当用户在选择日期文本框中输入所选择的日期后，选择日期文本框失去焦点时调用 date\_onchange 函数。该函数首先判断用户是否在选择日期文本框中输入有效日期，如果输入日期无效，则调用 setToday 函数在选择日期文本框与显示本日日期的 span 元素中显示本日日期，同时在日程提醒簿中显示用户之前在 localStorage 中保存的本日事件；如果输入日期有效，则在用来显示本日日期的 span 元素中显示用户所选择的日期，同时在日程提醒簿中显示用户之前在 localStorage 中保存的所选择日期的事件。

#### □ save 函数

单击保存按钮时调用 save 函数，在该函数中首先使用 obj 定义一个 JavaScript 对象，设定该对象有一个 record 属性，属性值为一个 JavaScript 数组，然后顺序读入页面中事件列表中每一个列表项目中的文字内容，如果该列表项目的文字内容不为“（尚未记录）”，则将该内容保存到 record 属性值数组中，最后将 obj 对象保存到 localStorage 中，键名为用户所选择的日期。

## 6.2 案例 18：临时保存页面中的输入内容

### 6.2.1 案例概述

在使用 Web 应用程序的时候，如果向页面中输入的内容比较多，有时可能在未输完全部内容时需要临时关闭浏览器（如突然有急事外出），这时我们希望能够在不将输入内容提交到服务器的前提下临时保存输入数据。本案例向大家展示如何实现这个功能。在本案例中，使用 localStorage 将用户输入内容保存在客户端计算机中，这样当用户下一次打开浏览器并访问案例页面时，页面中将自动显示用户保存在 localStorage 中的输入内容。

## 6.2.2 页面显示效果

本案例在第2章的案例4的基础上添加了将用户输入内容保存到 localStorage 中的功能，以及下次打开浏览器并访问本页面时页面中将自动显示用户保存在 localStorage 中的内容的功能。

首先，在案例页面中添加一个临时保存按钮，如图6-4所示（使用Google Chrome浏览器）。

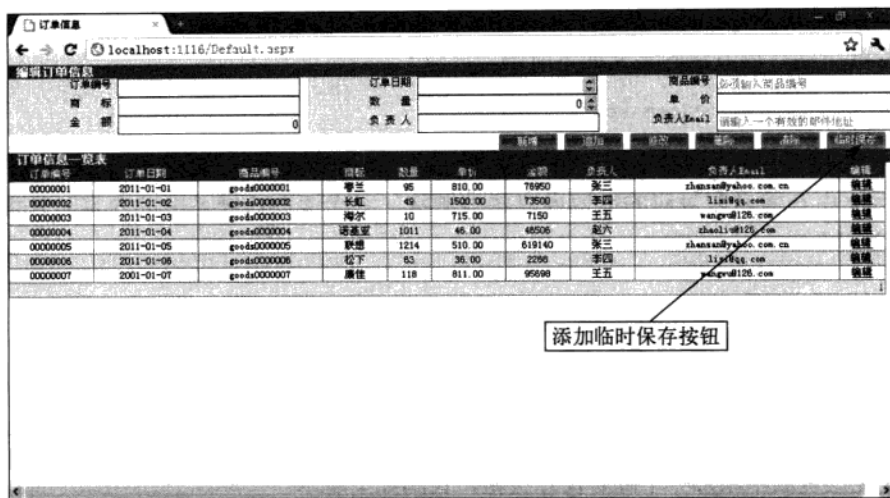


图 6-4 添加了临时保存按钮的案例页面

用户在该页面中输入内容并单击临时保存按钮，如图6-5所示，用户输入的内容将被保存在 localStorage 中。

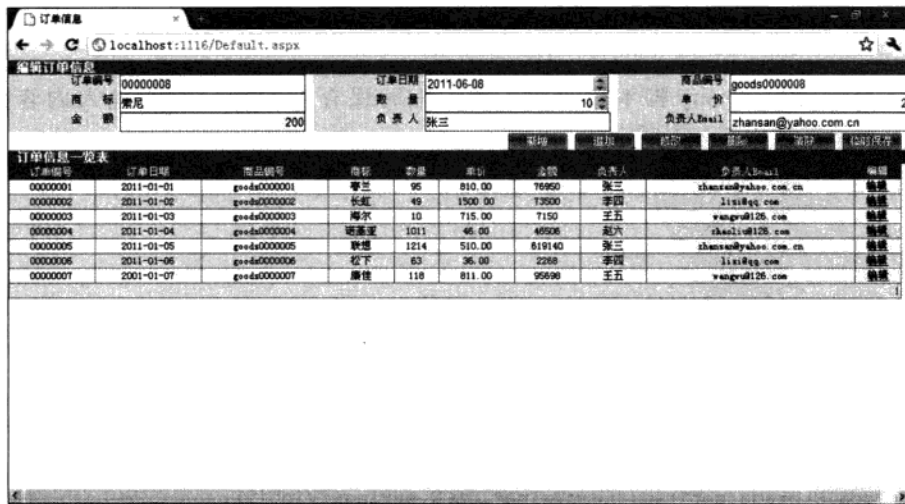


图 6-5 在该页面中输入内容并单击临时保存按钮



关闭浏览器后再次打开浏览器，然后访问本页面，页面将自动读取 localStorage 中的内容并将其显示在页面中，页面显示效果同图 6-5。

### 6.2.3 案例知识点

接下来看一下本案例中使用的 localStorage 对象的属性与方法。除了前面介绍过的 localStorage 对象的 getItem 方法与 setItem 方法外，本案例使用了 localStorage 对象的 removeItem 方法，该方法的定义如下所示。

```
localStorage.removeItem(key);
```

该方法接受一个参数 key，用该参数指定 localStorage 中的键名，执行该方法后 localStorage 中与键名相对应的键值内容将被删除，下次读取该键名中的内容时将读取不到（被设为 NULL）。

### 6.2.4 代码剖析

接下来看一下，为了实现将页面中的输入内容保存到 localStorage 中的功能及从 localStorage 中读取用户输入内容的功能，本案例在第 2 章的案例 4 的基础上进行了哪些必要的修改。

首先在该页面中添加一个临时保存按钮，代码如下所示。

```
<div id="buttonDiv">
    ... 页面中的其他按钮 ...
    <asp:Button ID="btnClear" runat="server" Text="清除" Enabled=false
    onclick="btnClear_Click"/>
    <!-- 将临时保存按钮添加在清除按钮的后面 -->
    <input type="button" id="btnSaveTemp" value="临时保存" />
</div>
```

接下来介绍在 jQuery 脚本代码中，单击临时保存按钮后将用户输入内容保存到 localStorage 中的代码，如下所示。

```
$("#btnSaveTemp").click(function () {
    var obj=new Object();
    obj.Code=$("#tbxCode").val();
    obj.Date=$("#tbxDate").val();
    obj.GoodsCode=$("#tbxGoodsCode").val();
    obj.BrandName=$("#tbxBrandName").val();
    obj.Num=$("#tbxNum").val();
    obj.Price=$("#tbxPrice").val();
    obj.Money=$("#tbxMoney").val();
    obj.PersonName=$("#tbxPersonName").val();
    obj.Email=$("#tbxEmail").val();
    localStorage.setItem("tempValue",JSON.stringify(obj));
});
```

在这段代码中，首先使用 obj 变量定义一个 JSON 对象，然后将用户在页面各控件中的内容保存在该对象的各属性中，最后将该对象保存在 localStorage 中。

接下来将 localStorage 中的内容读取到页面上的各控件中。

首先，在服务器端定义一个公有全局变量 isPostBack，用来通知前端页面是首次装载进来的，还是通过本页面的提交后返回的。变量 isPostBack 的定义如下所示。

```
public bool isPostBack=false;
```

经页面提交后将变量值设为 true，代码如下所示。

```
if (Page.IsPostBack)
    isPostBack = true;
```

在前端 jQuery 脚本代码中，首先判断全局变量 isPostBack 的值，如果为 false，表示首次装载本页面，这时将 localStorage 中保存的内容读取到由一个变量 obj 所引用的 JavaScript 对象的各属性中，同时将该 JSON 对象各属性中的内容显示在页面的各控件中。如果全局变量 isPostBack 的值为 true，则表示页面是通过提交后返回的，这时使用 localStorage 对象的 removeItem 方法将 localStorage 中保存的用户输入内容进行删除。将 localStorage 中的内容读取到页面上各控件中的 jQuery 脚本代码如下所示。

```
<%if(!isPostBack)
{
    var obj=JSON.parse(localStorage.getItem("tempValue"));
    if(obj!=null)
    {
        $("#tbxCode").val(obj.Code);
        $("#tbxDate").val(obj.Date);
        $("#tbxGoodsCode").val(obj.GoodsCode);
        $("#tbxBrandName").val(obj.BrandName);
        $("#tbxNum").val(obj.Num);
        $("#tbxPrice").val(obj.Price);
        $("#tbxMoney").val(obj.Money);
        $("#tbxPersonName").val(obj.PersonName);
        $("#tbxE-mail").val(obj.E-mail);
    }
}
else
{
    localStorage.removeItem("tempValue");
}
}>
```

## 6.3 案例 19：使用 HTML 5 制作 Web 应用程序的演示版

### 6.3.1 案例概述

本案例为一个使用 HTML 5 制作 Web 应用程序的演示版，在这个案例中，在不搭建 Web 服务器以及数据库环境的情况下，制作一个 Web 应用程序的演示版。在 HTML 5 之前，如果要制作这种 Web 应用程序的演示版，开发者将页面中的展示数据直接书写在页面中，因此不能很好地体现用户与 Web 应用程序演示版之间的交互性。如果使用 HTML 5，在不搭建 Web 服务器以及数据库环境的情况下，依靠 HTML 5 中 Web Database 的功能，将用户输入的数据保存在客户端的本地数据库（一种被称之为 SQLite 的文件型 SQL 数据库，以下称为 SQLite 数据库）中，这样演示版程序中对服务器端数据库的各种操作就能够依靠对 SQLite 数据库的各种操作体现出来，真正起到 Web 应用程序演示版的作用。

### 6.3.2 页面显示效果

首先来看一下案例页面在浏览器中的显示效果。本案例的页面非常类似于第 2 章中案例 4 的页面，都是一个用来输入订单信息的订单输入页面，用户在页面上半部分的表单中输入订单信息，保存后该订单信息将在页面下半部分的一览表中显示出来。同时，本案例页面中所使用的元素及其 HTML 页面代码也几乎与案例 4 完全相同，唯一区别为将服务器端按钮控件改为直接使用客户端按钮控件（input type=button 或 input type=submit）。

打开页面时一览表中将显示用户在本地数据库中保存的全部订单信息，如图 6-6 所示（使用 Google Chrome 浏览器）。

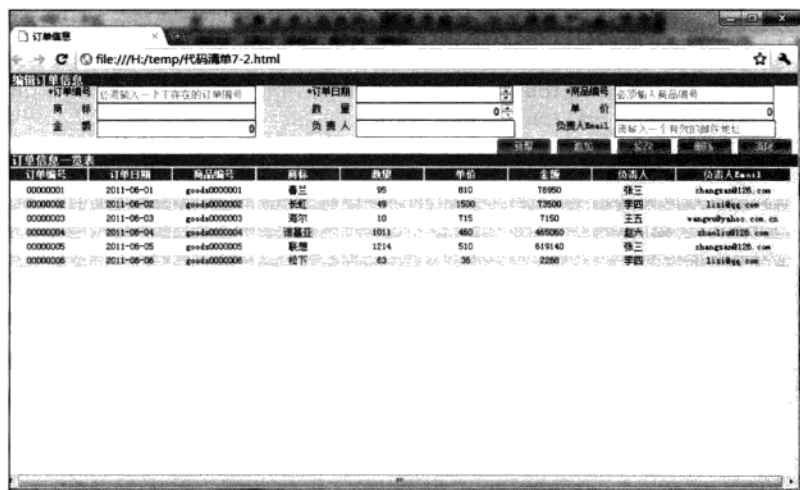


图 6-6 打开页面时一览表中显示本地数据库中保存的全部订单信息

用户可以在页面上半部分的表单中输入订单信息，如图 6-7 所示。

订单信息

file:///H:/temp/代码清单7-2.html

编辑订单信息

\*订单编号: 00000007 \*订单日期: 2011-06-07 \*商品编号: goods0000007

商 标: 康佳 数 量: 118 单 价: 811

金 额: 95696 负 责 人: 王五 负责人Email: wangwu@yahoo.com.cn

订单信息一览表

订单编号	订单日期	商品编号	商 标	数量	单价	金额	负责人	负责人Email
00000001	2011-06-01	goods0000001	格兰	95	810	76950	张三	zhangsan@126.com
00000002	2011-06-02	goods0000002	长虹	49	1500	73500	李四	lisi@qq.com
00000003	2011-06-03	goods0000003	海尔	10	715	7150	王五	wangwu@yahoo.com.cn
00000004	2011-06-04	goods0000004	诺基亚	1014	490	496060	赵六	zhao6@126.com
00000005	2011-06-05	goods0000005	联想	1214	510	619140	张三	zhangsan@126.com
00000006	2011-06-06	goods0000006	松下	63	36	2268	李四	lisi@qq.com

图 6-7 用户在表单中输入订单信息

输入订单信息后用单击保存按钮，该条信息将被保存到 SQLite 数据库中，同时在页面下半部的一览表中将该条信息显示出来，如图 6-8 所示。

订单信息

file:///H:/temp/代码清单7-2.html

编辑订单信息

\*订单编号: 00000007 \*订单日期: 2011-06-07 \*商品编号: goods0000007

商 标: 康佳 数 量: 118 单 价: 811

金 额: 95696 负 责 人: 王五 负责人Email: wangwu@yahoo.com.cn

订单信息一览表

订单编号	订单日期	商品编号	商 标	数量	单价	金额	负责人	负责人Email
00000001	2011-06-01	goods0000001	格兰	95	810	76950	张三	zhangsan@126.com
00000002	2011-06-02	goods0000002	长虹	49	1500	73500	李四	lisi@qq.com
00000003	2011-06-03	goods0000003	海尔	10	715	7150	王五	wangwu@yahoo.com.cn
00000004	2011-06-04	goods0000004	诺基亚	1014	490	496060	赵六	zhao6@126.com
00000005	2011-06-05	goods0000005	联想	1214	510	619140	张三	zhangsan@126.com
00000006	2011-06-06	goods0000006	松下	63	36	2268	李四	lisi@qq.com
00000007	2011-06-07	goods0000007	康佳	118	811	95696	王五	wangwu@yahoo.com.cn

被保存的数据

图 6-8 单击保存按钮后订单信息被显示在一览表中

### 6.3.3 案例知识点

接下来看一下本案例所使用的 Web Database 的一些属性与方法。关于 Web Database 的详细介绍，请参阅笔者所著的《HTML 5 与 CSS 3 权威指南》一书。

### □ openDatabase 方法

要使用 SQLite 数据库，首先必须使用 openDatabase 方法创建一个访问数据库的对象。该方法的使用如下所示。

```
var db = openDatabase('mydb', '1.0', 'Test DB', 2 * 1024 * 1024);
```

该方法有 4 个参数，第一个参数为数据库名，第二个参数为版本号，第三个参数为数据库的描述，第四个参数为数据库的大小，该方法返回创建后的数据库访问对象，如果该数据库不存在，则创建该数据库。

### □ transaction 方法

在实际操作数据库的时候，还需要调用 transaction 方法来执行事务处理。使用事务处理，可以防止在对数据库进行访问及执行有关操作时受到外界的打扰。因为在 Web 上，会有许多人同时对页面进行访问。如果在访问数据库的过程中，正在操作的数据被其他用户修改，会引起很多意想不到的后果。使用事务可以达到在操作完成之前，阻止其他用户访问数据库的目的。

transaction 方法的使用如下所示。

```
db.transaction(function (tx) {
    tx.executeSql('CREATE TABLE IF NOT EXISTS myTable (id unique, name TEXT)');
});
```

transaction 方法以一个回调函数为参数。在这个函数中，执行访问数据库的语句。

### □ executeSql 方法

在操作数据库的时候，还需要使用作为参数传递给回调函数的 transaction 对象的 executeSql 方法。

executeSql 方法的完整定义如下所示。

```
transaction.executeSql(sqlquery, [], dataHandler, errorHandler);
```

该方法有 4 个参数，第一个参数为需要执行的 SQL 语句。

第二个参数为 SQL 语句中所有使用的参数的数组。在 executeSql 方法中，将 SQL 语句中要使用的参数先用 ? 代替，然后依次将这些参数组成数组放到第二个参数中，代码类似如下所示。

```
transaction.executeSql("UPDATE myTable set field1=? where field2=?;", [value1, value2 ]);
```

第三个参数为执行 SQL 语句成功时调用的回调函数。该回调函数的传递方法如下所示。

```
function dataHandler(transaction, results){// 执行 SQL 语句成功时的处理};
```

该回调函数有两个参数，第一个参数为 transaction 对象，第二个参数为执行查询操作时返回的查询到的结果数据集对象。

第四个参数为执行 SQL 语句出错时调用的回调函数。该回调函数的传递方法如下所示。

```
function errorHandler(transaction,errmsg) { // 执行 SQL 语句出错时的处理 };
```

该回调函数有两个参数，第一个参数为 transaction 对象，第二个参数为执行发生错误时的错误信息文字。

#### □ 追加数据

追加数据时的代码如下所示。

```
tx.executeSql(
    'INSERT INTO myTable VALUES(?, ?, ?)',
    [value1, value2, value3],
    function(tx, rs) { .....追加数据成功时执行的处理..... },
    function(tx, error) { .....追加数据失败时执行的处理..... }
);
```

在这里，使用了“INSERT INTO MsgData VALUES(?, ?, ?)”这句 SQL 语句来追加数据，同时使用了 [value1, value2, value3] 数组来传入 SQL 语句中所需的参数。真正对数据库执行的 SQL 语句如下所示。

```
INSERT INTO myTable VALUES(value1 参数的值, value2 参数的值, value3 参数的值);
```

这条语句的作用是在数据表中插入一条数据。

#### □ 修改数据

修改数据时的代码如下所示。

```
tx.executeSql(
    'update myTable set field1=?,field2=?,field3=? where field4=?',
    [value1, value2,value3,value4],
    function(tx, rs) { .....修改数据成功时执行的处理..... },
    function(tx, error) { .....修改数据失败时执行的处理..... }
);
```

在这里，使用了“update myTable set field1=?,field2=?,field3=? where field4=?”这句 SQL 语句来追加数据，同时使用 [value1,value2,value3,value4] 数组来传入 SQL 语句中所需的参数。真正对数据库执行的 SQL 语句如下所示。

```
update myTable set field1=value1 参数的值,field2=value2 参数的值,
field3=value3 参数的值 where field4=value4 参数的值
```

这条语句的作用是在数据表中修改一条或多条数据（修改条件为“field4 字段的值 =value4 参数的值”）。

#### □ 删除数据

删除数据时的代码如下所示。

```
tx.executeSql(
    'delete from myTable where field=?',
    [value],
```

```
function(tx, rs) {……删除数据成功时执行的处理……},
function(tx, error) {……删除数据失败时执行的处理……}
);
```

在这里，使用了“delete from myTable where field=?”这句 SQL 语句来删除数据，同时使用 [value] 数组来传入 SQL 语句中所需的参数。真正对数据库执行的 SQL 语句如下所示。

```
delete from myTable where field=value 参数的值
```

这条语句的作用是在数据表中删除一条或多条数据（删除条件为“field 字段的值 =value 参数的值”）。

#### □ 创建数据表

创建数据表时的代码如下所示。

```
tx.executeSql(
    'CREATE TABLE IF NOT EXISTS myTable (
        field1 TEXT,
        field2 TEXT,
        field3 INTEGER)',
    []);
```

真正对数据库执行的 SQL 语句如下所示。

```
CREATE TABLE IF NOT EXISTS myTable(field1 TEXT, field2 TEXT, field3 INTEGER);
```

这条语句的作用是在数据库中创建一张数据表。

---

**注意** 如果已经存在数据表，重复创建该数据表会引发错误，所以前面必须要加上“IF NOT EXISTS”条件判断语句。这样，当要创建的表在数据库中已经存在时，就不会重复创建了。

---

#### □ 获取全部数据

获取全部数据的代码如下所示。

```
tx.executeSql('SELECT * FROM myTable', [], ……略…… );
```

这里，使用了“SELECT \* FROM myTable”这句 SQL 语句，该语句的作用是从 myTable 这张数据表中获取全部数据。

## 6.3.4 代码剖析

### 1. HTML 页面代码与样式代码

本案例页面中的 HTML 页面代码和样式代码与第 2 章的案例 4 中的 HTML 页面代码和样式代码几乎完全相同，区别为删除了第 2 章案例 4 中所有表单元元素的 class 属性，直接使用客户端的按钮控件，而不是使用服务器端的按钮控件。修改后的有关表单的 HTML 页面代码如下所示。







另外, 本案例不再使用第2章案例4中所使用的 JQuery 验证器, 而直接使用浏览器自带的验证功能, 故删除以下几行代码。

```
<link rel="stylesheet" href="Styles/validationEngine.jquery.css"
type="text/css" media="screen"/>
<script src="Scripts/jquery-1.5.1.min.js" type="text/javascript">
</script>
<script src="Scripts/jquery.validationEngine-cn.js"
type="text/javascript"></script>
<script src="Scripts/jquery.validationEngine.js" type="text/javascript">
</script>
```

本案例中其他 HTML 页面代码与样式代码均与第2章案例4中的 HTML 页面代码和样式代码完全相同, 故不再赘述。

## 2. JavaScript 脚本代码

接下来看一下本案例中所使用的 JavaScript 脚本代码。首先在代码清单 6-3 中列出本案例所使用的完整的 JavaScript 脚本代码, 稍后将对这份脚本代码进行详细分析。

代码清单 6-3 本案例所使用的完整的 JavaScript 脚本代码

---

```
<script>
var data = new Object;
var datatable;
var db = openDatabase('MyData', '', 'test Database', 102400);
function window_onload()
{
    datatable= document.getElementById("datatable");
    showAllData(true);
}
function tbxNum_onblur()
{
    var num,price;
    num=parseInt(document.getElementById("tbxNum").value);
    price=parseFloat(document.getElementById("tbxPrice").value);
    if (isNaN(num*price))
    {
        document.getElementById("tbxNum").value="0";
        document.getElementById("tbxMoney").value="0";
    }
    else
        document.getElementById("tbxMoney").value= num * price;
}

function tbxPrice_onblur()
{
    var num,price;
    num=parseInt(document.getElementById("tbxNum").value);
    price=parseFloat(document.getElementById("tbxPrice").value);
```

```

if (isNaN(num*price))
{
    document.getElementById("tbxPrice").value="0";
    document.getElementById("tbxMoney").value="0";
}
else
    document.getElementById("tbxMoney").value= num * price;
}
function btnAdd_onclick()
{
    data.Code=document.getElementById("tbxCode").value;
    data.Date=document.getElementById("tbxDate").value;
    data.GoodsCode=document.getElementById("tbxGoodsCode").value;
    data.BrandName=document.getElementById("tbxBrandName").value;
    data.Num=document.getElementById("tbxNum").value;
    data.Price=document.getElementById("tbxPrice").value;
    data.PersonName=document.getElementById("tbxPersonName").value;
    data.Email=document.getElementById("tbxEmail").value;
    db.transaction(function(tx)
    {
        tx.executeSql('CREATE TABLE IF NOT EXISTS orders(code TEXT,
        date TEXT,goodscode TEXT,brandName TEXT,num INTEGER,
        price FLOAT,personName TEXT,email TEXT)', []);
        tx.executeSql('SELECT * FROM orders where code=?', [data.Code],
        function(tx, rs)
        {
            if(rs.rows.length>0)
            {
                alert("输入的订单编号在数据库中已存在!");
                return;
            }
            tx.executeSql('INSERT INTO orders VALUES(?,?,?,?,?,?,?,?)',
            [data.Code,data.Date,data.GoodsCode,data.BrandName,data.Num,
            data.Price,data.PersonName,data.Email],
            function(tx, rs)
            {
                alert("成功保存数据!");
                showAllData(false);
                btnNew_onclick();
            },
            function(tx, error)
            {
                alert(error.source + "::" + error.message);
            });
        },
        function(tx, error)
        {
            alert(error.source + "::" + error.message);
        });
    });
}

```

```

    });
}
function btnUpdate_onclick()
{
    data.Code=document.getElementById("tbxCode").value;
    data.Date=document.getElementById("tbxDate").value;
    data.GoodsCode=document.getElementById("tbxGoodsCode").value;
    data.BrandName=document.getElementById("tbxBrandName").value;
    data.Num=document.getElementById("tbxNum").value;
    data.Price=document.getElementById("tbxPrice").value;
    data.PersonName=document.getElementById("tbxPersonName").value;
    data.Email=document.getElementById("tbxEmail").value;
    db.transaction(function(tx)
    {
        tx.executeSql('update orders set date=?,goodscode=?,brandName=?,
num=?,price=?,personName=?,email=? where code=?',
[data.Date,data.GoodsCode,data.BrandName, data.Num,data.Price,
data.PersonName,data.Email,data.Code],
function(tx, rs)
{
    alert(" 成功修改数据!");
    showAllData(false);
},
function(tx, error)
{
    alert(error.source + "::" + error.message);
});
});
}
function btnDelete_onclick()
{
    data.Code=document.getElementById("tbxCode").value;
    db.transaction(function(tx)
    {
        tx.executeSql('delete from orders where code=?',[data.Code],
function(tx, rs)
{
    alert(" 成功删除数据!");
    showAllData(false);
    btnNew_onclick();
},
function(tx, error)
{
    alert(error.source + "::" + error.message);
});
});
}
function btnNew_onclick()
{
    document.getElementById("form1").reset();
}

```

```

        document.getElementById("tbxCode").removeAttribute("readonly");
        document.getElementById("btnAdd").disabled="";
        document.getElementById("btnUpdate").disabled="disabled";
        document.getElementById("btnDelete").disabled="disabled";
    }
    function btnClear_onclick()
    {
        document.getElementById("tbxDate").value="";
        document.getElementById("tbxGoodsCode").value="";
        document.getElementById("tbxBrandName").value="";
        document.getElementById("tbxNum").value="0";
        document.getElementById("tbxPrice").value="0";
        document.getElementById("tbxMoney").value="0";
        document.getElementById("tbxPersonName").value="";
        document.getElementById("tbxEmail").value="";
    }
    function tr_onclick(tr,i)
    {
        document.getElementById("tbxCode").value=
            tr.children.item(0).innerHTML;
        document.getElementById("tbxDate").value=
            tr.children.item(1).innerHTML;
        document.getElementById("tbxGoodsCode").value=
            tr.children.item(2).innerHTML;
        document.getElementById("tbxBrandName").value=
            tr.children.item(3).innerHTML;
        document.getElementById("tbxNum").value=
            tr.children.item(4).innerHTML;
        document.getElementById("tbxPrice").value=
            tr.children.item(5).innerHTML;
        document.getElementById("tbxMoney").value=
            tr.children.item(6).innerHTML;
        document.getElementById("tbxPersonName").value=
            tr.children.item(7).innerHTML;
        document.getElementById("tbxEmail").value=
            tr.children.item(8).innerHTML;
        document.getElementById("tbxCode").setAttribute("readonly",true);
        document.getElementById("btnAdd").disabled="disabled";
        document.getElementById("btnUpdate").disabled="";
        document.getElementById("btnDelete").disabled="";
    }
    function showAllData(loadPage)
    {
        db.transaction(function(tx)
        {
            tx.executeSql('SELECT * FROM orders ', [], function(tx, rs)
            {
                if(!loadPage)
                    removeAllData();
                for(var i = 0; i < rs.rows.length; i++)

```

```

        {
            showData(rs.rows.item(i),i);
        }
    },
    function(tx, error)
    {
        alert(error.source + "::" + error.message);
    });
});
}
function removeAllData()
{
    for (var i =datatable.childNodes.length-1; i>1; i--)
        datatable.removeChild(datatable.childNodes[i]);
}
function showData(row,i)
{
    var tr = document.createElement('tr');
    tr.setAttribute("onclick", "tr_onclick(this, "+i+"");
    var td1 = document.createElement('td');
    td1.innerHTML = row.code;
    var td2 = document.createElement('td');
    td2.innerHTML = row.date;
    var td3 = document.createElement('td');
    td3.innerHTML = row.goodscode;
    var td4 = document.createElement('td');
    td4.innerHTML = row.brandName;
    var td5 = document.createElement('td');
    td5.innerHTML = row.num;
    var td6 = document.createElement('td');
    td6.innerHTML = row.price;
    var td7 = document.createElement('td');
    td7.innerHTML = parseInt(row.num)*parseFloat(row.price);
    var td8 = document.createElement('td');
    td8.innerHTML = row.personName;
    var td9= document.createElement('td');
    td9.innerHTML = row.email;
    tr.appendChild(td1);
    tr.appendChild(td2);
    tr.appendChild(td3);
    tr.appendChild(td4);
    tr.appendChild(td5);
    tr.appendChild(td6);
    tr.appendChild(td7);
    tr.appendChild(td8);
    tr.appendChild(td9);
    datatable.appendChild(tr);
}
</script>

```

在这份脚本代码的开始处，定义了本案例所使用的几个全局变量，其中 data 全局变量为一个 JavaScript 对象，在该对象中使用各种属性来保存用户在表单的各控件中输入的全部数据。datatable 全局变量表示页面下半部分的订单信息一览表，db 表示本案例中使用的本地 SQLite 数据库，在脚本的开始处使用 openDatabase 方法打开该数据库，数据库名为 MyData，不设置数据库版本号，数据库的描述为“test Database”，大小为 100K（102 400）。

#### ❑ window\_onload 函数

页面打开时调用 window\_onload 函数，调用代码如下所示。

```
<body onload="window_onload()">
```

该函数首先获取页面下半部分的订单信息一览表所使用的 table 元素并赋值给 datatable 全局变量，然后调用 showAllData 函数在订单信息一览表中显示用户之前在本地数据库中所保存的全部订单数据。

#### ❑ showAllData 函数

showAllData 函数的作用为在订单信息一览表中显示用户在本地数据库中所保存的全部订单数据。该函数接受一个参数 loadPage，该参数值为 true 表示本函数在页面打开时被调用，不需要删除订单信息一览表中当前显示的全部数据；该参数值为 false 表示本函数在用户追加或修改订单数据后被调用，需要先删除订单信息一览表中当前显示的全部数据，然后重新显示用户追加或修改订单数据后本地数据库中所保存的全部订单数据。

在 showAllData 函数中首先从本地数据库中获取全部订单数据，如果获取出错则弹出错误信息，如果获取成功后则判断 loadPage 参数值是否为 false，若是 false 则调用 removeAllData 函数删除订单信息一览表中当前显示的全部数据。获取全部订单数据（并删除订单信息一览表中当前显示的全部数据）后，循环调用 showData 函数在订单信息一览表中显示用户在本地数据库中所保存的全部订单数据（showData 函数的作用为在订单信息一览表中显示一条订单数据）。

#### ❑ removeAllData 函数

removeAllData 函数的作用为删除订单信息一览表中当前显示的全部数据，在函数中对订单信息一览表进行遍历，删除一览表中的每一行。

#### ❑ showData 函数

showData 函数的作用为在订单信息一览表中显示一条订单数据，该函数接受两个参数，row 参数代表用户保存的全部订单数据中的一行数据。案例程序获取到本地数据库中所保存的全部订单数据后，将这些数据以 rs.rows.item(i) 的形式作为 showData 函数的 row 参数传入 showData 函数中进行显示。rs.rows 代表获取到的数据的所有行，而 rs.rows.item(i) 则代表第 i 行中的数据，这些数据都以属性和属性值的形式存放在 rs.rows.item(i) 对象中，通过访问属性的方法来获取每个字段的内容。showData 函数的另一个参数 i 则代表了传入的行是所有订单数据中的第几行。

在 showData 函数的开始处，使用 tr 变量创建一个 tr 行元素（该元素代表一个 table 表元素中的一行），同时设定单击该行时调用 tr\_onclick 元素将被单击行中的所有列数据放入

表单内各控件中进行显示。创建完行元素后，创建 9 个列元素，并且在每个列元素中分别显示全部订单数据中传入行的订单数据中的订单编号、订单日期、商品编号、商标、数量、单价、金额、负责人、负责人 Email 信息，并且将这 9 个列元素添加到在函数开始处所创建的 tr 行元素中，同时将这个 tr 行元素添加到订单信息一览表中。

#### □ tr\_onclick 函数

单击订单信息一览表中某一行时将调用该函数。该函数的作用为将单击行中的所有列数据放入表单内各控件中进行显示。该函数接受两个参数，tr 参数代表一览表中被单击的 tr 行元素，i 参数代表被单击行的行号。在函数内部将表单中各元素的内容设置为被单击行的每一列的数据，同时将订单编号文本输入框设定为只读状态，将追加按钮设定为无效状态，将修改按钮与删除按钮设定为有效状态。

#### □ tbxNum\_onblur 函数

当光标焦点离开数量文本框时调用 tbxNum\_onblur 函数。该函数首先将数量文本框中的输入值转换为 int 类型后赋值给 num 变量，将单价文本框中的输入值转换为 float 类型后赋值给 price 变量，如果 num 变量值乘以 price 变量值后的结果不是有效数值，则将金额文本框中的内容设定为 0，否则（num 变量值乘以 price 变量值后的结果为有效数值）将金额文本框中的内容设定为 num 变量值乘以 price 变量值后的结果值。

#### □ tbxPrice\_onblur 函数

当光标焦点离开单价文本框时调用 tbxPrice\_onblur 函数。该函数首先将数量文本框中的输入值转换为 int 类型后赋值给 num 变量，将单价文本框中的输入值转换为 float 类型后赋值给 price 变量，如果 num 变量值乘以 price 变量值后的结果不是有效数值，则将单价文本框与金额文本框中的内容均设定为 0，否则（num 变量值乘以 price 变量值后的结果为有效数值）将金额文本框中的内容设定为 num 变量值乘以 price 变量值后的结果值。

#### □ btnAdd\_onclick 函数

用户单击追加按钮时调用 btnAdd\_onclick 函数。该函数首先将表单中各输入控件中的内容赋值给全局变量 data 的各个属性，然后判断本地数据库中是否存在 orders 数据表，如果不存在则创建该表，然后从 orders 数据表中选取订单编号等于用户输入的订单编号的记录，执行选取操作时如果发生错误将弹出错误提示信息，执行选取操作成功后如果数据表中存在该记录则弹出错误提示信息，内容为“输入的订单编号在数据库中已存在！”，然后退出函数，不执行插入操作。如果数据表中不存在订单编号等于用户输入的订单编号的记录，则在本地数据库中插入一条记录，记录中各字段的值为用户在表单的各控件中输入的值。插入时如果发生错误，则弹出错误提示信息。插入成功后将弹出提示信息，内容为“成功保存数据！”，然后调用 showAllData 函数在订单信息一览表中显示本地数据库中用户保存的所有订单信息，最后调用 btnNew\_onclick 函数将表单中各元素内容清空。

#### □ btnUpdate\_onclick 函数

用户单击修改按钮时将调用 btnUpdate\_onclick 函数。该函数首先将表单中各输入控件中的内容赋值给全局变量 data 变量的各个属性，然后将数据表中订单编号等于用户选择的订单编



号的记录中的各字段值（除了订单编号的值）修改为表单内各输入控件中的值，如果修改时发生错误则弹出错误提示信息，修改成功后弹出提示信息，内容为“成功修改数据！”，然后调用 `showAllData` 函数在订单信息一览表中重新显示在本地数据库中保存的所有订单信息。

#### ❑ `btnDelete_onclick` 函数

用户单击删除按钮时调用 `btnDelete_onclick` 函数。该函数首先将用户选取的订单编号赋值给全局变量 `data` 的 `Code` 属性，然后在数据表中删除订单编号等于用户所选取的订单编号的记录，如果删除时发生错误则弹出错误提示信息，删除成功后弹出提示信息，内容为“成功删除数据！”，之后调用 `showAllData` 函数在订单信息一览表中显示在本地数据库中保存的所有订单信息，最后调用 `btnNew_onclick` 函数将表单中各元素内容清空。

#### ❑ `btnNew_onclick` 函数

用户单击新增按钮时调用 `btnNew_onclick` 函数，该函数的作用为将表单中各元素设定为新增一条数据时的状态。在该函数内部，首先调用表元素的 `reset` 方法将表单中各输入控件中的内容清空，然后设定订单编号文本框为非只读状态，追加按钮为有效状态，修改按钮与删除按钮为无效状态。

#### ❑ `btnClear_onclick` 函数

用户在修改数据的过程中，单击清除按钮将调用该函数。在该函数中将表单中除订单编号文本框之外的其他输入框中的内容清空。

## 6.4 案例 20：使用客户端 session

### 6.4.1 案例概述

这个案例主要向读者介绍如何使用 HTML 5 中新增的 Web Storage 中的 `sessionStorage` 来实现客户端的 session 功能。本案例依然制作一个 Web 应用程序的演示版，该 Web 应用程序的演示版与案例 19 所制作的 Web 应用程序的演示版非常类似，实现的功能都是对商业订单进行增删查改操作，但是本案例使用两个页面，第一个页面为检索页面，用户在该页面中执行在本地数据库中检索订单数据的功能。第二个页面为订单编辑页面，用户在检索页面中检索到订单数据后，从本地数据库中获取的符合检索条件的订单信息会显示在页面下部的订单一览表中，单击一览表中每行最后一列中的编辑按钮，将在浏览器中关闭订单检索页面，打开订单编辑页面，同时在订单编辑页面的表单各控件中显示用户在订单检索页面的订单信息一览表中所单击行的每一列中的数据。同时，本案例通过 `sessionStorage` 来实现把用户在订单信息一览表中所单击行中每一列的数据传递到订单编辑页面中的功能。

### 6.4.2 页面展示效果

本案例使用两个页面——订单检索页面与订单编辑页面。首先看一下订单检索页面在浏览器中的显示效果。

打开订单检索页面后浏览器中的显示效果如图 6-9 中所示（使用 Google Chrome 浏览器）。

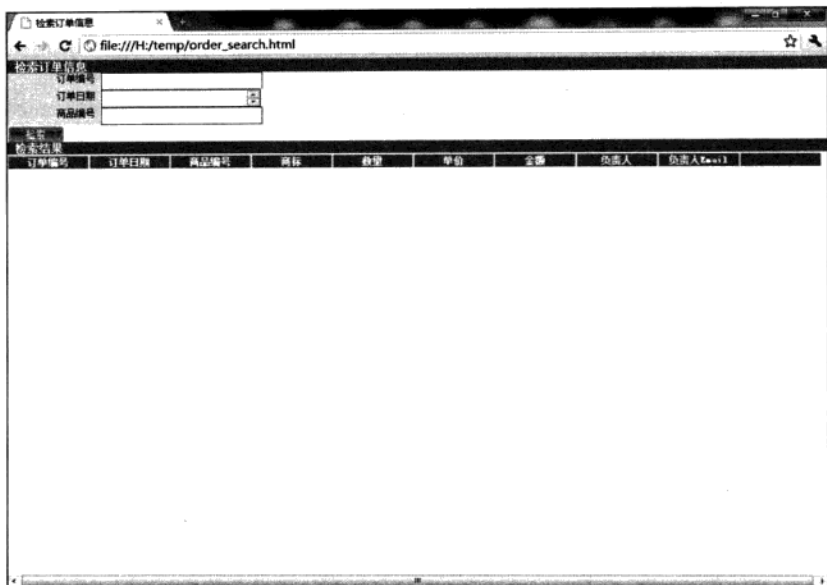


图 6-9 打开订单检索页面时的显示效果

用户可以在页面上半部分的订单编号文本框、订单日期文本框或商品编号文本框中输入检索条件，如图 6-10 所示。

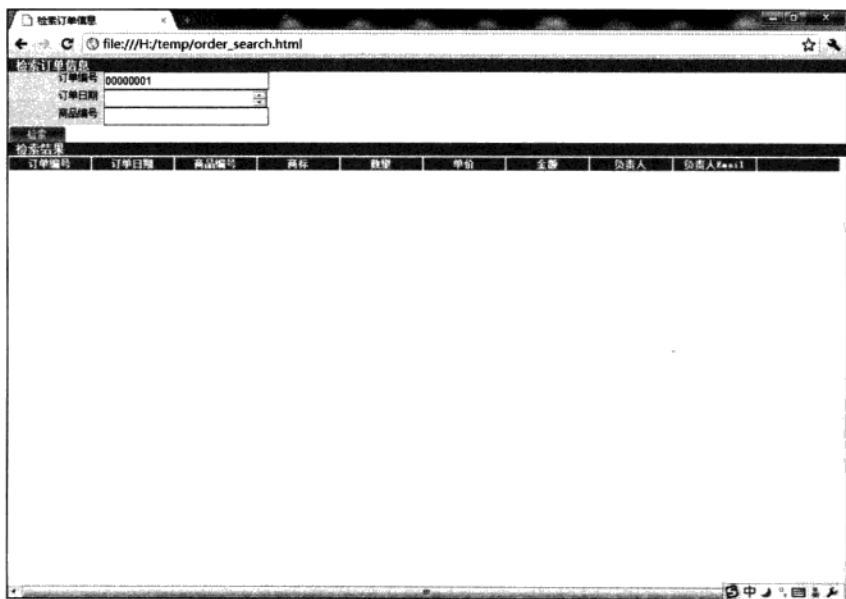


图 6-10 用户在订单编号文本框、订单日期文本框或商品编号文本框中输入检索条件

用户输入检索条件后单击检索按钮, 案例程序将在本地 SQLite 数据库中检索符合检索条件的记录, 检索到后将该记录显示在页面下半部分的检索结果一览表中, 如图 6-11 所示。

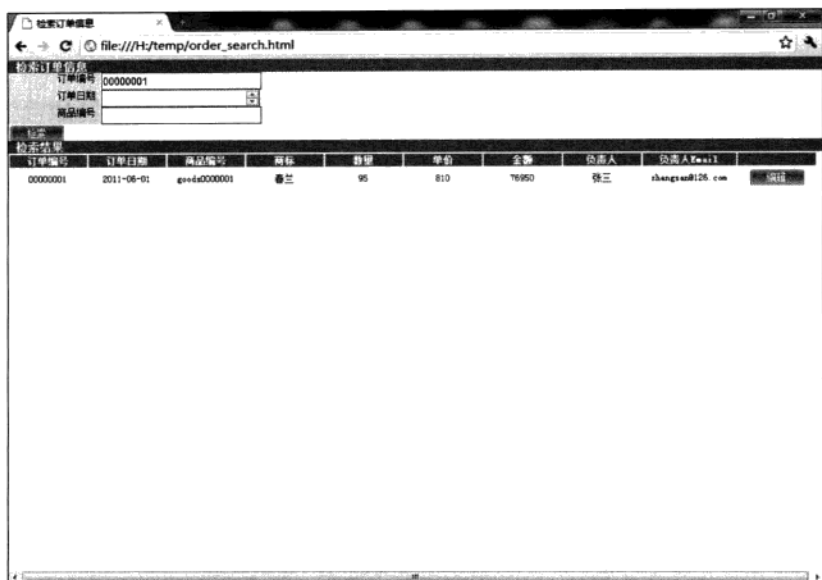


图 6-11 案例程序检索到记录后将结果显示在检索结果一览表中

用户单击每行最后一列中的编辑按钮, 浏览器中将关闭订单检索页面, 打开订单编辑页面。

接下来看一下订单编辑页面在浏览器中的显示效果。订单编辑页面在浏览器中打开时的显示效果如图 6-12 所示 (使用 Google Chrome 浏览器)。

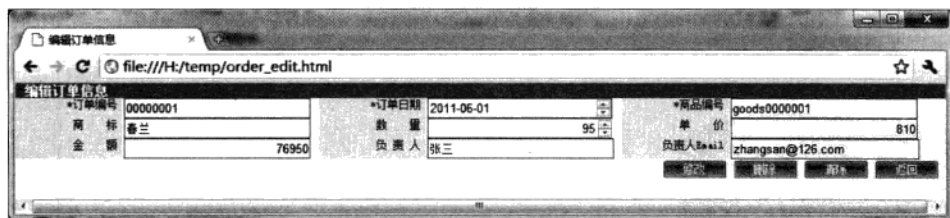


图 6-12 订单编辑页面在浏览器中打开时的显示效果

用户可以在该页面中修改或删除当前所选择的订单数据, 修改或删除该数据后单击页面中的返回按钮, 将在浏览器中关闭订单编辑页面, 重新打开订单检索页面。

### 6.4.3 案例知识点

接下来看一下本案例中使用到的 sessionStorage 的一些属性与方法。关于 sessionStorage 的详细介绍请参阅笔者所著的《HTML 5 与 CSS 3 权威指南》一书。

○ `sessionStorage.setItem (key,value)`

使用 `sessionStorage` 对象的 `setItem` 方法将数据保存在客户端的 `session` 中，保存数据时按“键名/键值”的形式进行保存。将第一个参数指定为键名，将第二个参数指定为键值。

保存时不允许重复保存相同的键名。保存后可以修改键值，但不允许修改键名（只能重新取键名，然后再保存键值）。

○ 变量 = `sessionStorage.getItem(key)`

使用 `sessionStorage` 对象的 `getItem` 方法读取 `session` 中的数据并保存到变量中，将参数指定为键名，返回键值并保存到变量中。

○ `sessionStorage.removeItem(key)`

使用 `sessionStorage` 对象的 `removeItem` 方法将客户端 `session` 中的指定键名所对应的键值数据进行删除。该方法接受一个参数 `key`，该参数指定 `sessionStorage` 中的键名。执行该方法后 `sessionStorage` 中与键名相对应的键值内容将被删除，下次读取该键名中的内容时将读取不到（被设为 `NULL`）。

## 6.4.4 代码剖析

### 1. HTML 页面代码

首先来看一下订单检索页面中各种元素的信息，如表 6-2 所示。

表 6-2 订单检索页面中各种元素的信息

控件名称	使用元素	显示文字	最大输入位数	说明
“检索订单信息”标题文字	h1	检索订单信息	—	
“订单编号”标签	label	订单编号	—	
“订单编号”文本框	input type="text"		8	
“订单日期”标签	label	订单日期	—	
“订单日期”文本框	input type="date"		10	
“商品编号”标签	label	商品编号	—	
“商品编号”文本框	input type="text"		12	
“订单信息一览表”标题文字	h5	订单信息一览表	—	
订单信息一览表	table			
“订单编号”列	数据列使用 td 元素， 标题列使用 th 元素			
“订单日期”列	数据列使用 td 元素， 标题列使用 th 元素			
“商品编号”列	数据列使用 td 元素， 标题列使用 th 元素			
“商标”列	数据列使用 td 元素， 标题列使用 th 元素			
“数量”列	数据列使用 td 元素， 标题列使用 th 元素			



```

<li id="title_3"><label for="tbxGoodsCode"> 商品编号 </label></li>
<li id="content_3">
  <input type="text" id="tbxGoodsCode" name="tbxGoodsCode"
    maxlength="12"/>
</li>
</ul>
</li>
</ul>
<div id="buttonDiv">
  <input type="submit" name="btnSearch" id="btnSearch" value=" 检索 "
    formaction="javascript:btnSearch_click();" />
</div>
</form>
<section>
<section>
<header id="div_head_title_big">
<h1> 检索结果 </h1>
</header>
<div id="infoTable">
<table id="datatable">
<tr>
  <th> 订单编号 </th>
  <th> 订单日期 </th>
  <th> 商品编号 </th>
  <th> 商标 </th>
  <th> 数量 </th>
  <th> 单价 </th>
  <th> 金额 </th>
  <th> 负责人 </th>
  <th> 负责人 Email </th>
  <th></th>
</tr>
</table>
</div>
</section>
</body>
</html>

```

订单检索页面中所使用的有关样式代码与案例 19 中所使用的样式代码基本相同，故不再赘述。

接下来看一下订单编辑页面中各种元素的信息，如表 6-3 所示。

表 6-3 订单编辑页面中各种元素的信息

控件名称	使用元素	显示文字	最大输入位数	说明
“编辑订单信息” 标题文字	h1	编辑订单信息	—	
“订单编号”标签	label	订单编号	—	

(续)

控件名称	使用元素	显示文字	最大输入位数	说明
“订单编号”文本框	input type="text"			编辑订单时该文本框为只读控件
“订单日期”标签	label	订单日期	—	
“订单日期”文本框	input type="date"		10	使用 placeholder 属性显示提示文字“必须输入一个有效的日期”。使用 required 属性在提交时必须输入验证
“商品编号”标签	label	商品编号	—	
“商品编号”文本框	input type="text"		12	使用 placeholder 属性显示提示文字“必须输入商品编号”。使用 required 属性在提交时必须输入验证
“商标”标签	label	商标		
“商标”文本框	input type="text"		50	
“数量”标签	label	数量		
“数量”文本框	input type="number"	画面打开时内容为“0”	6	使用 placeholder 属性显示提示文字“必须输入一个整数”
“单价”标签	label	单价		
“单价”文本框	input type="text"	画面打开时内容为“0”	6	使用 placeholder 属性显示提示文字“必须输入一个有效的单价”
“金额”标签	label	金额		
“金额”文本框	input type="text"	画面打开时内容为“0”		只读控件。当数量文本框或单价文本框失去焦点时设置“金额”文本框中内容 = 数量 × 单价
“负责人”标签	label	负责人		
“负责人”文本框	input type="text"		20	
“负责人 Email”标签	label	负责人 Email		
“负责人 Email”文本框	input type="email"		20	使用 placeholder 属性显示提示文字“请输入一个有效的邮件地址”
修改按钮	input type="submit"	修改		
删除按钮	input type="button"	删除		
清除按钮	input type="button"	清除		
返回按钮	input type="button"	返回		

订单编辑页面中的完整 HTML 页面代码如代码清单 6-5 所示。





[illegible]

```



```

订单编辑页面中所使用的有关样式代码与案例 19 中所使用的样式代码基本相同，故不再赘述。

## 2. JavaScript 脚本代码

接下来对本案例中所使用的 JavaScript 脚本代码进行详细分析。

订单检索页面中使用的全局变量及其说明与案例 19 的 JavaScript 脚本代码中使用的全局变量及其说明完全相同，故不再赘述。

本案例的订单检索页面的 JavaScript 脚本代码中使用如下函数。

### □ btnSearch\_click 函数

用户单击检索按钮时调用 btnSearch\_click 函数，函数代码如下所示。

```

function btnSearch_click()
{
    datatable= document.getElementById("datatable");
    data.Code=document.getElementById("tbxCode").value.trim();
    data.Date=document.getElementById("tbxDate").value.trim();
    data.GoodsCode=document.getElementById("tbxGoodsCode").value.trim();
    if (data.Code==" "&&data.Date==" "&&data.GoodsCode==" ")
        alert(" 必须输入一个检索条件 ");
    else
        searchData ();
}

```

btnSearch\_click 函数首先获取页面上的检索结果一览表中的元素并赋值给 datatable 全局变量。然后将订单编号文本框、订单日期文本框与商品编号文本框中的输入内容去除两边的空格后赋值给全局变量 data 对象中的 Code 属性、Date 属性与 GoodsCode 属性，如果三个文本框中的内容全为空白，则表示用户没有输入任何检索条件，这时弹出提示信息，提示用户必须输入检索条件。如果用户已输入检索条件，则调用 searchData 函数在本地 SQLite 数据库中检索符合检索条件的记录。

### □ searchData 函数

该函数的作用为在本地 SQLite 数据库中检索符合用户输入的检索条件的记录，代码如下所示。

```

function SearchData()
{
    db.transaction(function(tx)
    {
        var sql;
        var params=new Array();
        sql="SELECT * FROM orders where l=1";
        if(data.Code!="")
        {
            sql+=" and code=?";
            params.push(data.Code);
        }
        if(data.Date!="")
        {
            sql+=" and date=?";
            params.push(data.Date);
        }
        if(data.GoodsCode!="")
        {
            sql+=" and goodscode=?";
            params.push(data.GoodsCode);
        }
        tx.executeSql(sql,params, function(tx, rs)
        {
            removeAllData();
            for(var i = 0; i < rs.rows.length; i++)
            {
                showData(rs.rows.item(i),i);
            }
        },
        function(tx, error)
        {
            alert(error.source + "::" + error.message);
        });
    });
}

```

在这段代码中，首先书写在本地 SQL Lite 数据库中检索订单记录的 SQL 语句，接着判断全局变量 data 对象的 Code 属性值是否为空，如果不为空则表示用户在订单编号文本框中输入了订单编号，这时将用户输入的订单编号加入检索订单记录的 SQL 语句的检索条件中；然后判断全局变量 data 对象的 Date 属性的属性值是否为空，如果不为空则表示用户在订单日期文本框中输入了订单日期，这时将用户输入的订单日期加入检索订单记录的 SQL 语句的检索条件中；最后判断全局变量 data 对象的 GoodsCode 属性的属性值是否为空，如果不为空则表示用户在商品编号文本框中输入了商品编号，这时将用户输入的商品编号加入检索订单记录的 SQL 语句的检索条件中。书写完根据检索条件检索订单记录的 SQL 语句后在本地 SQLite 数据库中检索符合检索条件的记录，在检索过程中如果发生错误则弹出错误提示信息，检索成功后先调用 removeAllData 函数将检索结果一览表中当前显示的订单记录全部删

除, 然后循环调用 showData 函数在检索结果一览表中显示检索到的全部订单记录。

#### ❑ removeAllData 函数

removeAllData 函数的作用为将检索结果一览表中当前显示的订单记录全部删除, 函数代码及说明与案例 19 中所使用的 JavaScript 脚本代码中的 removeAllData 函数的代码及说明完全相同, 故不再赘述。

#### ❑ showData 函数

showData 函数的作用为在订单信息一览表中显示一条订单数据, 该函数接受两个参数, row 参数代表用户保存的全部订单数据中的一行数据。在案例程序检索到符合条件的订单数据后, 这些数据将以 rs.rows.item(i) 的形式作为 showData 函数的 row 参数传入到 showData 函数中进行显示。rs.rows 代表检索到的数据的所有行, 而 rs.rows.item(i) 则代表第 i 行中的数据, 这些数据都以属性和属性值的形式存放在 rs.rows.item(i) 对象中, 通过访问属性的方法来获取每个字段的内容。showData 函数的另一个参数 i 则代表传入的行是所有订单数据中的第几行。

showData 函数中的代码如下所示。

```
function showData(row,i)
{
    var tr = document.createElement('tr');
    var td1 = document.createElement('td');
    td1.innerHTML = row.code;
    var td2 = document.createElement('td');
    td2.innerHTML = row.date;
    var td3 = document.createElement('td');
    td3.innerHTML = row.goodscode;
    var td4 = document.createElement('td');
    td4.innerHTML = row.brandName;
    var td5 = document.createElement('td');
    td5.innerHTML = row.num;
    var td6 = document.createElement('td');
    td6.innerHTML = row.price;
    var td7 = document.createElement('td');
    td7.innerHTML = parseInt(row.num)*parseFloat(row.price);
    var td8 = document.createElement('td');
    td8.innerHTML = row.personName;
    var td9= document.createElement('td');
    td9.innerHTML = row.email;
    var td10= document.createElement('td');
    var btnEdit=document.createElement('button');
    btnEdit.innerHTML=" 编辑 ";
    btnEdit.setAttribute("onclick","btnEdit_click(this)");
    td10.appendChild(btnEdit);
    tr.appendChild(td1);
    tr.appendChild(td2);
    tr.appendChild(td3);
    tr.appendChild(td4);
```

```

tr.appendChild(td5);
tr.appendChild(td6);
tr.appendChild(td7);
tr.appendChild(td8);
tr.appendChild(td9);
tr.appendChild(td10);
datatable.appendChild(tr);
}

```

在 showData 函数的开始处, 使用变量 tr 创建了一个 tr 行元素 (该元素代表一个 table 表中的一行), 同时设定单击该行时将调用 tr\_onclick 元素将被单击行的所有列数据放入表单内各控件中进行显示。创建完行元素后, 创建 10 个列元素, 并且在每个列元素中分别显示检索到的订单记录中传入行的订单记录中的订单编号、订单日期、商品编号、商标、数量、单价、金额、负责人、负责人 Email 信息, 同时在最后一列中创建一个编辑按钮, 并且指定单击编辑按钮时调用 btnEdit\_click 函数关闭订单检索页面, 打开订单编辑页面, 然后将这 10 个列元素添加到在函数的开始处所创建的 tr 行元素中, 同时将这个 tr 行元素添加到订单信息一览表中。

#### □ btnEdit\_click 函数

用户单击检索结果一览表中某一行中的编辑按钮时调用 btnEdit\_click 函数, 函数代码如下所示。

```

function btnEdit_click(btnEdit)
{
    var tr = btnEdit.parentElement.parentElement;
    data = new Object();
    data.Code=tr.cells[0].innerHTML;
    data.Date=tr.cells[1].innerHTML;
    data.GoodsCode=tr.cells[2].innerHTML;
    data.BrandName=tr.cells[3].innerHTML;
    data.Num=tr.cells[4].innerHTML;
    data.Price=tr.cells[5].innerHTML;
    data.Money=tr.cells[6].innerHTML;
    data.PersonName=tr.cells[7].innerHTML;
    data.Email=tr.cells[8].innerHTML;
    var str = JSON.stringify(data);
    sessionStorage.setItem("saveData",str);
    window.location=setLocation();
}

```

该函数接受一个参数 btnEdit, 该参数代表了被单击的编辑按钮。

在函数内部中, 首先获取传入参数 btnEdit 元素的父元素, 即被单击的编辑按钮所在的行元素, 然后使用全局变量 data 创建一个 JSON 对象, 并将被单击行中的所有列数据赋值给 data 对象的各个属性, 接下来使用 JSON 对象的 stringify 方法将 data 对象转换为字符串, 之后使用 sessionStorage 对象的 setItem 方法将 data 对象转换成的字符串保存在客户端 session 中, 保存时的键名为 “saveData”, 最后调用 setLocation 方法获取访问订单编辑页面时的

URL 地址，并且将 window 对象的 location 属性的属性值设定为访问订单编辑页面时的 URL 地址（即执行关闭订单检索页面，打开订单编辑页面的操作）。

#### ❑ setLocation 函数

setLocation 函数的作用为返回访问订单编辑页面时的 URL 地址，函数代码如下所示。

```
function setLocation()
{
    var location=String(window.location);
    location=location.replace("search","edit");
    return location;
}
```

在该函数中，首先获取浏览器当前所使用的 URL 地址（即访问订单检索时的 URL 地址），因为笔者将订单检索页面与订单编辑页面放于同一目录中，且将订单检索页面命名为“order-search.html”，将订单编辑页面命名为“order-edit.html”，所以使用“location.replace("search","edit");”语句即可直接得到访问订单编辑页面时的 URL 地址，然后将该 URL 地址返回。

本案例的订单编辑页面的 JavaScript 脚本代码中使用如下全局变量与函数。

#### ❑ 全局变量定义

在订单编辑页面的 JavaScript 脚本代码的开始处，定义了本案例中所使用的几个全局变量，代码如下所示。

```
var data;
var db = openDatabase('MyData', '', 'test Database', 102400);
```

在这两句代码中，全局变量 data 为一个 JavaScript 对象，在该对象中使用各种属性来保存用户在表单的各控件中输入的全部数据。全局变量 db 代表本案例中使用的本地 SQLite 数据库，在代码中使用 openDatabase 方法打开该数据库，数据库名为 MyData，不设置数据库版本号，数据库的描述为“test Database”，大小为 100K（102 400）。

#### ❑ window\_onload 函数

页面打开时调用 window\_onload 函数，代码如下所示。

```
function window_onload()
{
    var str = sessionStorage.getItem("saveData");
    data = JSON.parse(str);
    document.getElementById("tbxCod").value=data.Code;
    document.getElementById("tbxDate").value=data.Date;
    document.getElementById("tbxGoodsCode").value=data.GoodsCode;
    document.getElementById("tbxBrandName").value=data.BrandName;
    document.getElementById("tbxNum").value=data.Num;
    document.getElementById("tbxPrice").value=data.Price;
    document.getElementById("tbxMoney").value=data.Money;
    document.getElementById("tbxPersonName").value=data.PersonName;
```

```

document.getElementById("tbxEmail").value=data.Email;
document.getElementById("tbxCode").setAttribute("readonly",true);
}

```

在这段代码中，使用 sessionStorage 对象的 getItem 方法获取客户端 session 中保存的 saveData 键名所对应的键值数据（在订单检索页面中该键值数据被保存为一个 JSON 对象，该对象的各属性中保存了用户在检索结果一览表中所选的订单记录的各字段值），然后将获取到的键值数据通过 JSON.parse 方法还原为 JSON 对象，再将该对象的各属性值显示在页面上表单的各控件中。

#### ❑ tbxNum\_onblur 函数

tbxNum\_onblur 函数的代码及说明与案例 19 中所使用的 JavaScript 脚本代码中的 tbxNum\_onblur 函数的代码及说明完全相同，故不再赘述。

#### ❑ tbxPrice\_onblur 函数

tbxPrice\_onblur 函数的代码及说明与案例 19 中所使用的 JavaScript 脚本代码中的 tbxPrice\_onblur 函数的代码及说明完全相同，故不再赘述。

#### ❑ btnUpdate\_onclick 函数

单击修改按钮时调用 btnUpdate\_onclick 函数将用户在表单中修改完毕后的订单数据更新到本地 SQLite 数据库中，函数代码及说明与案例 19 中所使用的 JavaScript 脚本代码中的 btnUpdate\_onclick 函数的代码及说明几乎完全相同，只是删除了案例 19 中调用 showAllData 函数在订单信息一览表中显示本地 SQLite 数据库中全部订单数据的代码，修改后的 btnUpdate\_onclick 函数的代码如下所示。

```

function btnUpdate_onclick()
{
    data.Code=document.getElementById("tbxCode").value;
    data.Date=document.getElementById("tbxDate").value;
    data.GoodsCode=document.getElementById("tbxGoodsCode").value;
    data.BrandName=document.getElementById("tbxBrandName").value;
    data.Num=document.getElementById("tbxNum").value;
    data.Price=document.getElementById("tbxPrice").value;
    data.PersonName=document.getElementById("tbxPersonName").value;
    data.Email=document.getElementById("tbxEmail").value;
    db.transaction(function(tx)
    {
        tx.executeSql('update orders set date=?,goodscode=?,brandName=?,
num=?,price=?,personName=?,email=? where code=?',
        [data.Date,data.GoodsCode,data.BrandName, data.Num,data.Price,
        data.PersonName,data.Email,data.Code],
        function(tx, rs)
        {
            alert("成功修改数据!");
        },
        function(tx, error)
        {

```

```

        alert(error.source + "::" + error.message);
    });
}

```

#### ❑ btnDelete\_onclick 函数

单击删除按钮时调用 btnDelete\_onclick 函数将用户在订单检索页面中所选择的订单记录进行删除，函数代码及说明与案例 19 中所使用的 JavaScript 脚本代码中的 btnDelete\_onclick 函数的代码及说明几乎完全相同，只是删除了案例 19 中调用 showAllData 函数在订单信息一览表中显示本地 SQLite 数据库中全部订单数据以及调用 btnNew\_onclick 将表单各元素内容进行清空的代码，添加了调用 btnReturn\_onclick 函数在删除记录后在浏览器中关闭订单编辑页面，打开订单检索页面的代码，修改后的 btnDelete\_onclick 函数的代码如下所示。

```

function btnDelete_onclick()
{
    data.Code=document.getElementById("tbxCode").value;
    db.transaction(function(tx)
    {
        tx.executeSql('delete from orders where code=?',[data.Code],
            function(tx, rs)
            {
                alert("成功删除数据!");
                btnReturn_onclick();
            },
            function(tx, error)
            {
                alert(error.source + "::" + error.message);
            });
    });
}

```

#### ❑ btnClear\_onclick 函数

单击清除按钮时调用 btnClear\_onclick 函数，函数代码及说明与案例 19 中所使用的 JavaScript 脚本代码中的 btnClear\_onclick 函数的代码及说明完全相同，故不再赘述。

#### ❑ btnReturn\_onclick 函数

单击返回按钮时调用 btnReturn\_onclick 函数，函数代码如下所示。

```

sessionStorage.removeItem("saveData");
window.location=setLocation();

```

btnReturn\_onclick 函数先调用 sessionStorage 对象的 removeItem 方法将客户端 session 中键名“saveData”所对应的键值删除，然后调用 setLocation 方法获取访问订单检索页面时的 URL 地址，并且将 window 对象的 location 属性值设定为访问订单检索页面时的 URL 地址（即执行关闭订单编辑页面，打开订单检索页面的操作）。



#### ❑ setLocation 函数

setLocation 函数的作用为返回访问订单检索页面时的 URL 地址，函数代码如下所示。

```
function setLocation()  
{  
    var location=String(window.location);  
    location=location.replace("edit","search");  
    return location;  
}
```

在该函数中，首先获取浏览器中当前所使用的 URL 地址（即访问订单编辑页面时的 URL 地址），因为笔者将订单检索页面与订单编辑页面放于同一目录中，且将订单检索页面命名为“order-search.html”，将订单编辑页面命名为“order-edit.html”，所以使用“location.replace("edit","search");”语句即可直接得到访问订单检索页面时的 URL 地址，然后将该 URL 地址返回。

## 6.5 案例 21：将本地数据库中的数据提交到服务器端

### 6.5.1 案例概述

很多时候需要将客户端本地数据库的数据提交到服务器端，然后在服务器端将这些数据保存到服务器端的数据库中。例如，商业网站中“将商品放入购物车”这一功能，用户（包括未登录网站的匿名用户）在挑选好商品之后通过单击“放入购物车”按钮将挑选好的商品放入购物车中，而在实现代码中将用户挑选好的商品的相关数据（例如货名、数量、单价、金额等信息）保存在本地数据库中，只有用户登录了网站并且单击“结算”按钮后才将用户本地数据库中的商品信息提交到服务器端并保存在服务器端的数据库中，同时生成订单信息，打开收货信息填写界面，供用户填写个人收货信息。将用户临时挑选的商品信息保存在客户端本地数据库中可以减少服务器端数据库资源的浪费（因为数据被分散保存在客户端本地，而且很多匿名用户只是临时保存一下，以后再也没有访问过这个商业网站），同时也在一定程度上提高了服务器端数据库的性能。

本案例向读者展示如何将客户端本地的 SQLite 数据库中的数据提交到服务器端的数据库中进行保存，以案例 19 中制作的 Web 应用程序的演示版为基础，在页面中添加一个提交按钮，单击该按钮后将本地 SQLite 数据库中保存的订单信息提交到服务器端并保存到数据库中（案例中使用 SQL Server 2005 Express 数据库）。

### 6.5.2 页面展示效果

本案例页面以案例 19 的页面为基础，添加了一个提交按钮，如图 6-13 所示（使用 Google Chrome 浏览器）。

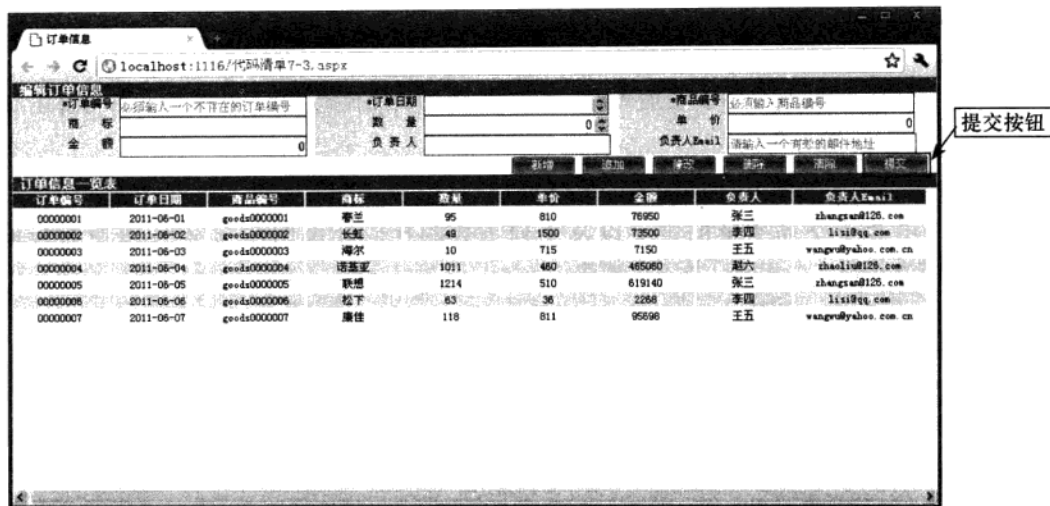


图 6-13 案例页面在打开时的显示效果

单击提交按钮后，本地 SQLite 数据库中的订单数据将被全部提交到服务器端并被保存到服务器端的数据库中，然后本地 SQLite 数据库中的订单数据将被全部删除。

### 6.5.3 代码剖析

#### 1. HTML 页面代码

本页面在案例 19 页面的基础上添加了一个提交按钮，同时添加了一个隐藏控件，当单击提交按钮把本地数据库中的数据提交到服务器端时将把本地数据库中的数据先转换成为一个由 JavaScript 对象组成的数组，之后使用 JSON 对象的 stringify 方法将这个数据转换为字符串，并将该字符串保存到该隐藏控件中，然后进行提交，后台服务器端代码将从这个隐藏控件中取出该字符串，再将该字符串还原成 JavaScript 对象的数组。

提交按钮及隐藏控件的 HTML 页面代码如下所示。

```
<div id="buttonDiv">
  <input type="button" name="btnNew" id="btnNew" value=" 新增 "
  onclick="btnNew_onclick();" />
  <input type="submit" name="btnAdd" id="btnAdd" value=" 追加 "
  formation="javascript:btnAdd_onclick();" />
  <input type="submit" name="btnUpdate" id="btnUpdate" value=" 修改 "
  disabled formation="javascript:btnUpdate_onclick();" />
  <input type="button" name="btnDelete" id="btnDelete" value=" 删除 "
  disabled onclick="btnDelete_onclick();" />
  <input type="button" name="btnClear" id="btnClear" value=" 清除 "
  onclick="btnClear_onclick();" />
  <input type="button" name="btnSubmit" id="btnSubmit" value=" 提交 "
  onclick="btnSubmit_onclick();" />
</div>
```

```

</div>
<input type="hidden" id="JSONText" name="JSONText"/>

```

本案例页面中的其他 HTML 页面代码与案例 19 中的页面代码完全相同，因此不再赘述。

## 2. JavaScript 脚本代码

接下来看一下本案例中的 JavaScript 脚本代码部分。

本案例中关于全局变量的定义以及下列函数的代码及说明均与案例 19 中相应的 JavaScript 脚本代码完全相同，故不再赘述。

- ❑ tbxNum\_onblur 函数
- ❑ tbxPrice\_onblur 函数
- ❑ btnAdd\_onclick 函数
- ❑ btnUpdate\_onclick 函数
- ❑ btnDelete\_onclick 函数
- ❑ btnNew\_onclick 函数
- ❑ btnClear\_onclick 函数
- ❑ tr\_onclick 函数
- ❑ showAllData 函数
- ❑ removeAllData 函数
- ❑ showData 函数

下面看一下单击提交按钮时所调用的 btnSubmit\_onclick 函数，代码如下所示。

```

function btnSubmit_onclick()
{
    datatable= document.getElementById("datatable");
    var tr;
    var dataArray=new Array();
    for(var i=2;i<datatable.childNodes.length;i++)
    {
        data = new Object();
        tr=datatable.childNodes[i];
        data.Code=tr.childNodes[0].innerHTML;
        data.Date=tr.cells[1].innerHTML;
        data.GoodsCode=tr.cells[2].innerHTML;
        data.BrandName=tr.cells[3].innerHTML;
        data.Num=tr.cells[4].innerHTML;
        data.Price=tr.cells[5].innerHTML;
        data.PersonName=tr.cells[7].innerHTML;
        data.Email=tr.cells[8].innerHTML;
        dataArray.push(JSON.stringify(data));
    }
    document.getElementById("JSONText").value=JSON.stringify(dataArray);
    form1.submit();
}

```

btnSubmit\_onclick 函数首先获取页面上的订单信息一览表元素，并把它赋值给全局变量 datatable，然后使用全局变量 dataArray 定义一个 JavaScript 数组。接下来，循环获取订单信息一览表中的所有数据（这时该一览表中显示本地数据库中的所有订单数据），在循环中使用变量 data 定义一个 JavaScript 对象，并将一览表中每一行的各列数据保存在 data 对象的各个属性中，然后使用 JSON 对象的 stringify 方法将 data 对象转换为字符串并保存在 dataArray 数组中，循环结束后再使用 JSON 对象的 stringify 方法将 dataArray 数组转换为字符串并保存在页面的隐藏控件中，最后将表单提交，提交表单后将利用服务器端的脚本代码来获取隐藏控件中的数据并将这些数据保存到服务器端的数据库中，服务器端的脚本代码文件为“代码清单 7-3.aspx”，指定方法如下所示（在关于表单定义的 HTML 页面代码中）。

```
<form id="form1" method="post" action=" 代码清单 7-3.aspx">
```

接下来看一下提交后重新打开本页面时所调用的 window\_onload 函数，代码如下所示。

```
function window_onload()
{
    datatable = document.getElementById("datatable");
    <%if (postFlag)
    {%>
        db.transaction(function (tx) {
            tx.executeSql('delete from orders', [],
                function (tx, rs) {
                    alert(" 成功提交数据，本地数据库中数据被清空！");
                },
                function (tx, error) {
                    alert(error.source + "：" + error.message);
                }
            );
        });
    <%}%>
    showAllData(true);
}
```

window\_onload 函数首先获取页面上的订单信息一览表元素，并把它赋值给全局变量 datatable，然后根据服务器端变量 postFlag 的值来判断页面是首次被装载还是在单击了提交按钮并从服务器端返回后的装载，如果是从服务器端返回后的装载，则删除本地 SQLite 数据库中的全部订单数据，删除产生错误时弹出错误提示信息，删除成功后弹出“成功提交数据，本地数据库中数据被清空！”的提示信息。不管页面是首次被装载还是在单击了提交按钮并从服务器端返回后的装载，都调用 showAllData 函数重新将本地 SQLite 数据库中的全部订单数据显示在页面的订单信息一览表中。

### 3. ASP.NET 服务器端脚本代码

接下来看一下代码清单 7-3.aspx 文件中的 ASP.NET 服务器端的脚本代码，如代码清单 6-6 所示。

代码清单 6-6 代码清单 7-3.aspx 文件中的 ASP.NET 服务器端的脚本代码

```

using System;
using System.Collections;
using System.Collections.Generic;
using System.Text;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.IO;
using System.Configuration;
using System.Data;
using System.Data.SqlClient;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Json;
namespace HTML5TEST
{
    public partial class 代码清单 7-3: System.Web.UI.Page
    {
        [DataContract]
        public class Data
        {
            [DataMember]
            public String Code { get; set; }
            [DataMember]
            public String Date { get; set; }
            [DataMember]
            public String GoodsCode { get; set; }
            [DataMember]
            public String BrandName { get; set; }
            [DataMember]
            public int Num { get; set; }
            [DataMember]
            public float Price { get; set; }
            [DataMember]
            public String PersonName { get; set; }
            [DataMember]
            public String Email { get; set; }
        }
        private System.Data.SqlClient.SqlConnection SqlCon;
        private System.Data.SqlClient.SqlCommand Command;
        private int myErrorNumber = 0;
        private string myErrorMessage = "";
        public bool postFlag = false;
        protected void Page_Load(object sender, EventArgs e)
        {
            string Constr, SqlStr;
            bool SuccessFlag;

            if (Page.IsPostBack)
            {

```

```

Constr = System.Configuration.ConfigurationManager.
ConnectionStrings["dbConnection"].ToString();
this.SqlCon = new System.Data.SqlClient.SqlConnection();
this.SqlCon.ConnectionString = Constr;
SuccessFlag = OpenDBConnection();
if (SuccessFlag == false)
{
    this.ShowError("数据库连接失败。");
    return;
}
String str = Request.Form["JSONText"].ToString();
byte[] buffer = Encoding.UTF8.GetBytes(str);
DataContractJsonSerializer serializerArray=
new DataContractJsonSerializer(typeof(ArrayList));
DataContractJsonSerializer serializer =
new DataContractJsonSerializer(typeof(Data));
using (MemoryStream stream = new MemoryStream(buffer))
{
    ArrayList dataArray =
serializerArray.ReadObject(stream) as ArrayList;
    for (int i = 0; i < dataArray.Count; i++)
    {
        str = dataArray[i].ToString();
        buffer = Encoding.UTF8.GetBytes(str);
        using (MemoryStream stream1 = new MemoryStream(buffer))
        {
            Data myData =
serializer.ReadObject(stream1) as Data;
            SqlStr = "insert into orders ";
            SqlStr += "values('" +
myData.Code.Trim().Replace("'", "'') + "','";
            SqlStr += "'" + myData.Date.Trim() + "','";
            SqlStr += "'" +
myData.GoodsCode.Trim().Replace("'", "'') + "','";
            SqlStr += "'" +
myData.BrandName.Trim().Replace("'", "'') + "','";
            SqlStr += myData.Num.ToString() + "','";
            SqlStr += myData.Price.ToString() + "','";
            SqlStr += "'" +
myData.PersonName.Trim().Replace("'", "'') + "','";
            SqlStr += "'" +
myData.Email.Trim().Replace("'", "'') + '');"
            SuccessFlag = this.ExecSingleSql(SqlStr);
            if (SuccessFlag == false)
            {
                this.ShowError("数据添加失败。");
                return;
            }
        }
        postFlag = true;
    }
}

```

```

        }
    }
}

private bool OpenDBConnection()
{
    try
    {
        this.SqlCon.Open();
        return true;
    }
    catch (SqlException e)
    {
        this.myErrorNumber = e.Number;
        this.myErrorMessage = e.Message;
        return false;
    }
    catch (Exception e)
    {
        this.myErrorNumber = e.GetHashCode();
        this.myErrorMessage = e.Message;
        return false;
    }
}

public void ShowError(string myErrorMessage)
{
    string strErrMsg;
    if (!this.myErrorMessage.Equals(""))
        strErrMsg = this.myErrorNumber + ":" + this.myErrorMessage;
    else
        strErrMsg = myErrorMessage;
    Page.ClientScript.RegisterStartupScript(Page.GetType(), "",
        "<script language='JavaScript'>alert('" + strErrMsg +
        "');</script>");
}

private bool ExecSingleSql(string SqlStr)
{
    try
    {
        this.Command = new SqlCommand(SqlStr);
        this.Command.Connection = this.SqlCon;
        this.Command.ExecuteNonQuery();
        return true;
    }
    catch (SqlException e)
    {
        this.myErrorNumber = e.Number;
        this.myErrorMessage = e.Message;
        return false;
    }
}

```

```

        catch (Exception e)
        {
            this.myErrorNumber = e.GetHashCode();
            this.myErrorMessage = e.Message;
            return false;
        }
    }
}

```

这段代码为 ASP.NET 服务器端脚本代码。对 ASP.NET 服务器端脚本代码的详细讲解不在本书所介绍的范围之中，这里只讲解这段代码的处理流程。

首先定义一个 Data 类，该类的作用类似于本案例中 JavaScript 脚本代码中的 Data 对象，每一个 Data 对象中存放一条订单数据，每个 Data 对象的各个属性值中存放该订单数据的各部分细节信息（订单号、订单日期和商品编号等）。

另外，为了能够将客户端传过来的 JSON 对象解析为服务器端对象，需要将 Data 对象的类型声明为 DataContract 类型，并且将 Data 对象的每一个属性声明为 DataMember 类型，这样才能将客户端的 Data 对象解析为服务器端的 Data 对象，并且能够获取客户端的 Data 对象中的完整数据。

接下来，在代码中声明了连接 SQL Server 数据库时所使用的 SqlCon 对象与对 SQL Server 数据库进行操作时所使用的 Command 对象，这些对象代表了对数据库的操作发生错误时所使用的错误号与错误信息的全局变量 myErrorNumber 与全局变量 myErrorMessage，以及向客户端表明是否已执行过将客户端数据库中的数据提交到服务器端这一处理的变量 postFlag。

当用户单击提交按钮将客户端数据库中的内容提交到服务器端时，服务器端将再次调用 Page\_Load 事件处理函数，并且将 Page.IsPostBack 置为 true，这时首先连接数据库，如果连接失败将弹出“数据库连接失败。”的错误提示信息。数据库连接成功后首先获取页面上隐藏控件中的内容（这时该隐藏控件中保存了一个由 JavaScript 对象的数组转换而来的字符串，数组中保存了所有由客户端数据库中所有订单数据生成的 JSON 对象）。然后声明了将客户端的数组解析为服务器端数组时所使用的 serializerArray 对象，将客户端的 JavaScript 对象解析为服务器端对象时所使用的 serializer 对象。接下来先将客户端传过来的由 Data 对象组成的数组解析为服务器端数组，然后对该数组进行遍历，将数组中的每一个客户端的 Data 对象解析为服务器端的 Data 类的对象，并将该对象中的数据插入到 SQL Server 数据库中，如果插入失败，则弹出“数据添加失败。”的错误提示信息。插入成功后将 postFlag 置为 true，向客户端表明数据已成功提交到服务器端的数据库中。



## 6.6 案例 22：制作可以离线使用的日程提醒簿

### 6.6.1 案例概述

本案例将对案例 17 制作的 HTML 5 版本的日程提醒簿添加离线功能，添加离线功能后用户可以在不与案例所在的网站建立网络连接的情况下继续使用这个日程提醒簿功能。通过这个案例来向读者介绍如何在 HTML 5 中制作离线 Web 应用程序。

### 6.6.2 页面显示效果

首先来看一下本案例页面在浏览器中的显示效果。本案例的页面显示效果与案例 17 的页面显示效果完全相同。不同之处仅在于添加了离线应用程序的功能，这样用户在第一次访问案例页面之后，案例所在网站会将该页面所使用的资源文件下载到客户端计算机（或移动设备）的本地缓存中，下一次即使客户端计算机（或移动设备）不与案例所在网站建立网络连接也可以继续使用案例中的日程提醒簿。一旦客户端计算机（或移动设备）再次与该网站建立连接，如果案例页面所使用的资源文件存在更新，客户端计算机（或移动设备）会再次将更新后的资源文件自动下载到客户端计算机（或移动设备）中，同时页面上弹出“本地缓存已被更新，需要刷新画面来获取应用程序最新版本，是否刷新？”的确认信息，如图 6-14 所示（使用 Google Chrome 浏览器）。



图 6-14 页面上弹出信息询问用户是否立即刷新页面来使用最新资源文件

如果用户单击确认信息中的确定按钮，则立即刷新页面，并使用最新的资源文件，否则资源文件将在用户手工刷新页面或下次访问该页面时被使用。

### 6.6.3 案例知识点

接下来介绍本案例中所使用的有关制作离线 Web 应用程序的一些知识点，关于离线 Web 应用程序的详细知识，请参阅笔者所著的《HTML 5 与 CSS 3 权威指南》一书。

#### 1. manifest 文件

Web 应用程序的本地缓存是通过每个页面的 manifest 文件来管理的。manifest 文件是一个简单文本文件，在该文件中以清单的形式列举了需要被缓存或不需要被缓存的资源文件的名称，以及这些资源文件的访问路径。可以为每一个页面单独指定一个 manifest 文件，也可以对整个 Web 应用程序指定一个总的 manifest 文件。代码清单 6-7 为 manifest 文件的一个示例，该文件为 hello.html 网页的 manifest 文件，通过这个示例对 manifest 文件进行详细介绍。

代码清单 6-7 manifest 文件示例

---

```
CACHE MANIFEST
# 文件的开头必须要书写 CACHE MANIFEST
# 这个 manifest 文件的版本号
#version 7
CACHE:
other.html
hello.js
images/myphoto.jpg
NETWORK:
http://Lulingniu/NotOffline
NotOffline.asp
*
FALLBACK:
online.js locale.js
CACHE:
newhello.html
newhello.js
```

---

在 manifest 文件中，第一行必须是“CACHE MANIFEST”，目的是把本文件的作用告知浏览器，即对本地缓存中的资源文件进行具体设置。同时，在真正运行或测试离线 Web 应用程序的时候，需要对服务器进行配置，使服务器支持 text/cache-manifest 这个 MIME 类型（在 HTML5 中规定 manifest 文件的 MIME 类型是 text/cache-manifest）。例如，在对 Apache 服务器进行配置的时候，需要找到 {apache\_home}/conf/mime.types 这个文件，并在文件最后添加如下所示的一行代码。

```
text/cache-manifest    manifest
```

在微软的 iis 服务器中找到 manifest 文件的步骤如下所示。

- 1) 选择默认网站或需要添加类型的网站，右击，弹出属性对话框。

- 2) 选择“HTTP 头”标签。
- 3) 在 MIME 映射下, 单击文件类型按钮。
- 4) 在打开的 MIME 类型对话框中单击新建按钮。
- 5) 在关联扩展名文本框中输入“manifest”, 在内容类型文本框中输入“text/cache-manifest”, 然后单击确定按钮。

在 manifest 文件中, 可以加上注释来进行一些必要的说明或解释, 注释行以“#”文字开头。

在 manifest 文件中可以(而且最好)加上一个版本号, 以表示这个 manifest 文件的版本。版本号可以是任何形式, 如上面的“version 201011211108”, 更新 manifest 文件时一般也要对这个版本号进行更新。

在指定资源文件的时候, 文件路径可以是相对路径, 也可以是绝对路径。指定时每个资源文件为一行。

指定资源文件时可以把资源文件分为三类, 分别是 CACHE、NETWORK、FALLBACK。

- ❑ 在 CACHE 类别中指定需要被缓存在本地的资源文件。为某个页面指定需要本地缓存的资源文件时, 不需要把这个页面本身指定在 CACHE 类别中, 因为如果一个页面具有 manifest 文件, 浏览器会自动对这个页面进行本地缓存。
- ❑ 在 NETWORK 类别中显式指定不进行本地缓存的资源文件, 这些资源文件只有当客户端与服务器端建立连接时才能被访问。本示例中该类别中的“\*”为通配符, 表示没有在本 manifest 文件中指定的资源文件都不进行本地缓存。
- ❑ 在 FALLBACK 类别的每行中指定两个资源文件, 第一个资源文件为能够在线访问时使用的资源文件, 第二个资源文件为不能在线访问时使用的备用资源文件。

每个类别都是可选的。但是如果在文件开头没有指定类别而直接书写资源文件, 浏览器会把这些资源文件视为 CACHE 类别, 直到看见文件中第一个被书写出来的类别为止。

为了让浏览器能够正常阅读该文本文件, 需要在 Web 应用程序页面的 html 标签的 manifest 属性中指定 manifest 文件的 URL 地址。指定方法如下所示。

```
<!--你可以为每个页面单独指定一个 manifest 文件-->
<html manifest="hello.manifest">
...
</html>
<!--也可以为整个 Web 应用程序指定一个总的 manifest 文件-->
<html manifest="global.manifest">
...
</html>
```

通过这些步骤, 将资源文件保存到本地缓存区的基本操作就完成了。当要对本地缓存区的内容进行修改时, 只要修改 manifest 文件就可以了。文件被修改后, 浏览器可以自动检查 manifest 文件, 并自动更新本地缓存区中的内容。

## 2. applicationCache 对象

applicationCache 对象代表了本地缓存，可以通过它来通知用户本地缓存已经被更新。

在浏览器与服务器的交互过程中，在浏览器对本地缓存进行更新并装入新的资源文件后，会触发 applicationCache 对象的 updateready 事件，通知本地缓存已被更新。可以利用这个事件告诉用户本地缓存已经被更新，用户需要手工刷新页面来得到最新版本的应用程序。这部分代码如下所示。

```
applicationCache.onUpdateReady = function () {
    // 本地缓存已被更新，通知用户
    alert("本地缓存已被更新，您可以刷新页面来得到本程序的最新版本。");
};
```

另外，可以通过 applicationCache 的 swapCache 方法来控制如何进行本地缓存的更新以及更新的时机。

## 3. swapCache 方法

用户可以通过调用 swapCache 方法来手工执行对本地缓存的更新，这个方法只能在 applicationCache 对象的 updateReady 事件被触发时被调用，updateReady 事件只有在服务器上的 manifest 文件被更新，并且把 manifest 文件中所要求的资源文件下载到本地后触发。顾名思义，updateReady 事件的含义是“本地缓存准备被更新”。当这个事件被触发后，可以利用 swapCache 方法来手工更新本地缓存。下面来看一下哪些场合需要使用到这个方法。

首先，如果本地缓存的容量非常大（譬如超过 100MB），本地缓存的更新工作将需要相对较长的时间，而且还会把浏览器锁住。这时最好有一个提示，告诉用户正在进行本地缓存的更新，该部分代码如下所示。

```
applicationCache.onUpdateReady = function () {
    // 本地缓存已被更新，通知用户
    alert("正在更新本地缓存……");
    applicationCache.swapCache();
    alert("本地缓存已被更新，您可以刷新页面来得到本程序的最新版本。");
};
```

这时考虑一个问题，在上面的代码中，如果不调用 swapCache 方法会怎么样？本地缓存就不会被更新了吗？回答是否定的，但是，更新的时间不一样。如果不调用 swapCache 方法，本地缓存将在下一次打开本页面时被更新；如果调用 swapCache 方法，本地缓存将会被立刻更新。因此，可以使用 confirm 方法让用户自己选择更新的时机——是立刻更新，还是在下次打开画面时再更新，特别是正在页面上执行一个较大的操作的时候。

另外，尽管使用 swapCache 方法会立刻更新本地缓存，但是这并不意味着页面上的图像和脚本文件也会被立刻更新，它们都是在重新打开本页面时才会生效的。

## 6.6.4 代码剖析

### 1. HTML 页面代码与样式代码

本案例中的 HTML 页面代码与样式代码与案例 17 中的 HTML 页面代码与样式代码基本相同，不同之处仅在于将 JavaScript 脚本提到了一个单独的 script.js 文件之中，在页面中将原来的 JavaScript 脚本代码改成为引用外部 JavaScript 脚本文件，代码如下所示。

```
<script src="script.js"></script>
```

另外，在页面开头处使用作为缓存清单的 manifest 文件（文件名为 book.manifest），代码如下所示。

```
<!DOCTYPE html>
<html manifest="book.manifest">
```

### 2. 缓存清单

接下来看一下本案例中使用的缓存清单文件，文件名为 book.manifest 文件，文件中的内容如下所示。

```
CACHE MANIFEST
#version 1.0
CACHE:
script.js
日历背景.png
```

缓存清单的最初版本为 1.0 版本，今后每次修改客户端本地缓存中使用的资源文件时都需要更新版本号，这样客户端计算机（或移动设备）再次与本网站建立连接并访问案例页面时才能将修改过的资源文件下载到客户端计算机（或移动设备）中。另外，需要指定在客户端计算机（或移动设备）中缓存案例页面所使用的样式文件与背景图片。

### 3. JavaScript 脚本代码

最后来看一下本案例所使用的脚本文件 script.js 中的脚本代码，该脚本代码中的 date\_onchange 函数、save 函数、setInnerHTML 函数、setToday 函数均与案例 17 的 JavaScript 代码中的这几个函数完全相同，故此处不再赘述。下面介绍打开页面时所调用的 window\_onload 函数。

```
function window_onload() {
    dateElement=document.getElementById("date1");
    today=document.getElementById("today");
    setToday();
    setInterval(function() {
        // 手工检查是否有更新
        applicationCache.update();
    }, 5000);
    applicationCache.addEventListener("updateready", function() {
```

```

if (confirm("本地缓存已被更新，需要刷新画面来获取应用程序最新版本，是否刷新？")) {
    // (3) 手工更新本地缓存
    applicationCache.swapCache();
    // 重载画面
    location.reload();
}
}, true);
}

```

该函数仍然是先获取页面中的选择日期文本框并将其赋值给 `dateElement` 全局变量，获取页面中用来显示本日日期的 `span` 元素并将其赋值给 `today` 全局变量，然后调用 `setToday` 函数将本日日期显示在日期文本框与用来显示本日日期的 `span` 元素中，将用户保存的本日要处理的事件显示在日程提醒簿中。

接下来，设置每 5 秒钟自动检查一下服务器端的资源文件是否被更新（服务器端的资源文件被修改后会自动下载到客户端计算机或移动设备中），一旦当服务器端的资源文件被更新并且被下载到客户端计算机（或移动设备）中，会弹出提示信息，询问用户是否立即更新本地缓存，如果用户单击确定按钮，客户端计算机（或移动设备）中的本地缓存将被立即更新，同时页面被刷新，页面中使用更新后的资源文件；否则（用户单击取消按钮）本地缓存将在案例页面被刷新或下一次打开案例页面时被更新。

## 6.7 本章小结

本章通过几个案例来具体阐述 HTML 5 中本地存储的概念及其使用方法，共涉及三个内容：Web Storage、Web Database 与离线 Web 应用程序的开发。希望读者通过本章的阅读，能够对于本地存储的知识有一个完整的了解，能够在自己的 Web 网站或 Web 应用程序中正确使用 Web Storage 与 Web Database，能够开发出自己的离线 Web 应用程序。

下一章将详细阐述 HTML 5 中跨文档消息传输功能的基本知识及使用方法，通过案例来介绍如何通过跨文档消息传输功能来实现多个 Web 系统中的单点登录以及从一个 Web 系统中获取另一个 Web 系统中的批量数据。



## 第7章

# 跨文档消息传输

### 本章内容

- ☐ 案例 23：通过跨文档消息传输功能实现单点登录
- ☐ 案例 24：通过跨文档消息传输功能获取批量数据
- ☐ 本章小结

本章通过两个案例来详细阐述 HTML 5 中的跨文档消息传输功能及使用方法。案例 23 将介绍如何使用跨文档消息传输功能来实现多个 Web 系统中的单点登录功能，案例 24 将介绍如何使用跨文档消息传输功能来实现在一个网站中获取另一个网站中所使用的一批数据并将其显示在页面上。

## 7.1 案例 23：通过跨文档消息传输功能实现单点登录

### 7.1.1 案例概述

本节通过一个案例来介绍如何通过 HTML 5 中的跨文档消息传输功能来实现多个 Web 系统中的单点登录，在详细介绍该案例前，首先需要向部分对单点登录不大熟悉的读者解释一下单点登录的基本概念。

#### 1. 单点登录的基本概念

进入 21 世纪之后，随着网络应用的普及，越来越多的大中型企业要求员工每天通过 Web 系统来进行日常工作。在一个企业中，一般有 OA 系统、经销存系统、财务系统、客户关系管理系统、决策支持系统等，由于安全需要（判断用户是否能进入本系统）及权限需要（判断用户进入本系统之后能进行哪些操作，不能进行哪些操作），用户在进入每一个系统时，都需要提供一个用户名及密码，因此用户需要记住很多用户名和密码，这样会给用户带来不少麻烦，特别是随着系统的增多，出错的可能性就会增加，受到非法截获和破坏的可能性也会增大，安全性就会相应降低许多。为解决此类愈显突出的安全问题，采用单点登录技术来提高企业中 Web 系统的安全性显得尤为重要。

单点登录（Single Sign-On, SSO），是一种 Web 系统的用户管理机制。用户在网络中只需要主动地进行一次身份认证鉴别登录，即可进入各种需要访问的 Web 系统，并获得由系统管理员指定的各种操作权限。

#### 2. 单点登录所需实现的功能

具体来说，要实现单点登录，需要实现以下主要的功能：

□ 所有应用系统共享一个身份认证系统。

统一的认证系统是单点登录的前提之一。认证系统的主要功能是将用户的登录信息和用户信息库相比较，对用户进行登录认证；认证成功后，认证系统应该生成统一的认证标志（ticket），返还给用户。另外，认证系统还应该对 ticket 进行校验，判断其有效性。

□ 所有应用系统能够识别和提取 ticket 信息。

要实现单点登录的功能，使用户只登录一次即可访问需要访问的 Web 系统，就必须使应用系统能够识别已经登录过的用户。应用系统应该能对 ticket 进行识别和提取，通过与认证系统的通信，自动判断当前用户是否登录过，从而完成单点登录的功能。

另外，有以下两点需要注意：



□ 单一的用户信息数据库并不是必需的。

有许多系统不能将所有的用户信息都集中存储，应该允许将用户信息放置在不同的存储中。事实上，只要统一认证系统，统一 ticket 的产生和校验，无论用户信息存储在什么地方，都能实现单点登录。

□ 统一的认证系统并不只有单个的认证服务器。

认证服务器之间要通过标准的通信协议互相交换认证信息，才能完成更高级别的单点登录。例如，当用户访问应用系统 1 时，经第一个认证服务器认证后，得到由此服务器产生的 ticket，当他访问应用系统 2 时，认证服务器 2 能够识别此 ticket 是由第一个服务器产生的，通过认证服务器之间标准的通信协议（如 SAML）来交换认证信息，能够完成 SSO 的功能。

### 3. 案例程序所实现的功能

在本案例中，将向读者阐述如何通过 HTML 5 中的跨文档消息传输功能来实现单点登录。案例中有三个系统，一个用户身份统一认证系统、一个用 Java 语言中的 struts 2 框架 + spring 2 框架 + hibernate 3 框架（以下简称 ssh 框架）开发出来的仅供本案例测试用的简易 Web 系统与一个用 ASP.NET 语言开发出来的供 HTML 5 中各种 API（包括本书前面所介绍的一些 API）测试时用的 Web 系统。用户可以在用户身份统一认证系统的登录页面中输入用户名与密码，然后单击该页面中的登录按钮，系统同时将用户名与密码发送给 Java 版本的 Web 系统与 ASP.NET 版本的 Web 系统，这两个系统在接收到用户身份统一认证系统传递过来的用户名与密码后自动实现登录操作，如果使用接收到的用户名与密码登录失败，则弹出错误提示信息，提示用户登录失败。同时，用户也可分别在这两个 Web 系统的登录页面中输入用户名与密码，如果在一个 Web 系统中登录成功后，将把用户名与密码发送到另一个 Web 系统中，另一个 Web 系统在接收到用户名与密码后，将自动实现登录操作，如果使用该用户名与密码登录失败，则弹出错误提示信息，提示用户登录失败。虽然在本案中实现单点登录的前提是，用户在 Java 版本的 Web 系统中进行登录时所使用的用户名与密码与用户在 ASP.NET 版本的 Web 系统中进行登录时所使用的用户名与密码必须完全相同，但是只要理解了本节中介绍的使用跨文档消息传输功能实现单点登录操作的原理，并稍微扩展一下，就可以让用户在 Java 版本的 Web 系统中使用一套用户名与密码，在 ASP.NET 版本的 Web 系统中使用另一套用户名与密码，仍然能够实现单点登录操作。这一个扩展工作由读者自行完成，在本章小结中，我们将对扩展方法进行详细阐述。

## 7.1.2 页面显示效果

### 1. Java 版本的 Web 系统的页面显示效果

首先来看一下本案例中通过 Java 语言中的 ssh 框架开发的用来测试跨文档消息传输功能的 Web 系统中的各页面在浏览器中的显示效果（使用 Google Chrome 浏览器）。

网站首页的登录页面在浏览器中的显示效果，如图 7-1 所示（注意，访问的 URL 地址为 <http://localhost:8443/HTML5/>）。

用户可以在该页面中输入用户名与密码，如图 7-2 所示。

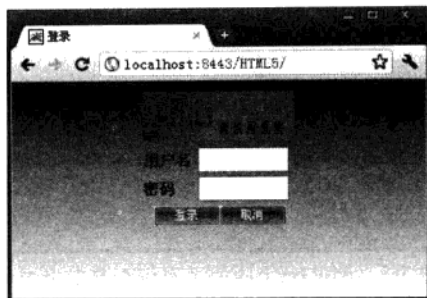


图 7-1 Java 版测试系统的登录页面

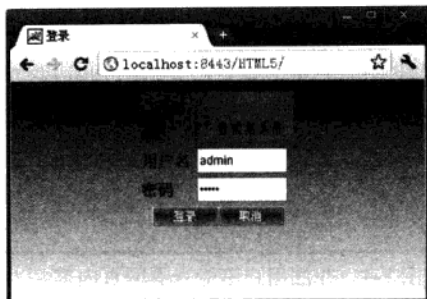


图 7-2 用户在登录页面中输入用户名与密码

输入用户名与密码后单击登录按钮，登录成功后浏览器中的登录页面被关闭，打开欢迎页面，同时向 ASP.NET 版本的测试系统发送用户名与密码，ASP.NET 版本的测试系统在接收到用户名与密码后，自动实现登录操作，登录成功后页面上弹出提示信息，提示用户成功登录 ASP.NET 版本的测试系统，如图 7-3 所示（欢迎页面仅作测试之用，故只简单显示一下用户名）。



图 7-3 成功登录 ASP.NET 版本的测试系统

在 Java 版本的测试系统中，访问欢迎页面时所使用的 URL 地址为“http://localhost:8443/HTML5//welcome.action?crossDomainFlag=”，如果用户不是在首页上输入并提交用户名与密码而是直接访问该页面地址（或 http://localhost:8443/HTML5//welcome.action?crossDomainFlag=true，或 http://localhost:8443/HTML5//welcome.action?crossDomainFlag=false），或直接打开页面文件 welcome.jsp，页面上将显示一片空白，以防止非法用户查看网页内容，如图 7-4 所示。

## 2. ASP.NET 版本的 Web 系统的页面显示效果

接下来看一下本案例中通过 ASP.NET 语言开发出来的供跨域通信测试所用的 Web 系统中的各页面在浏览器中的显示效果（使用 Google Chrome 浏览器）。

登录页面在浏览器中的显示效果如图 7-5 所示（注意，访问该页面时的 URL 地址为 http://localhost:1116/index.aspx）。

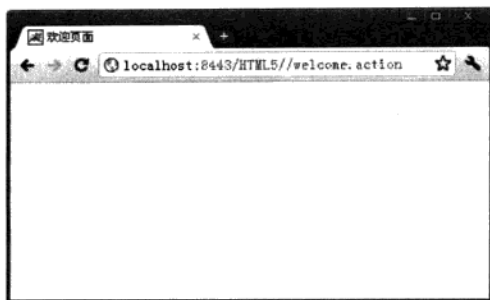


图 7-4 非法用户不能查看网页内容

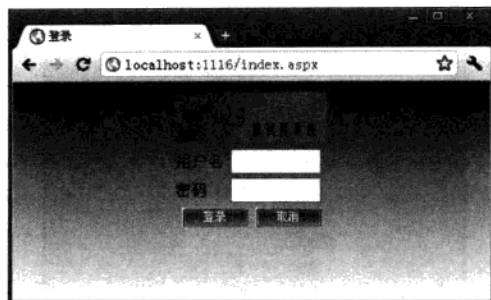


图 7-5 ASP.NET 版测试系统的登录页面

用户输入用户名与密码后单击登录按钮，登录成功后浏览器中的登录页面被关闭，打开本书第 2 章案例 4 中制作的案例页面，在该页面中显示用户名，同时向 Java 版本的测试系统发送用户名与密码，Java 版本的测试系统在接收到用户名与密码后，自动实现登录操作，登录成功后页面上弹出提示信息，提示用户成功登录 Java 版本的测试系统，如图 7-6 所示。

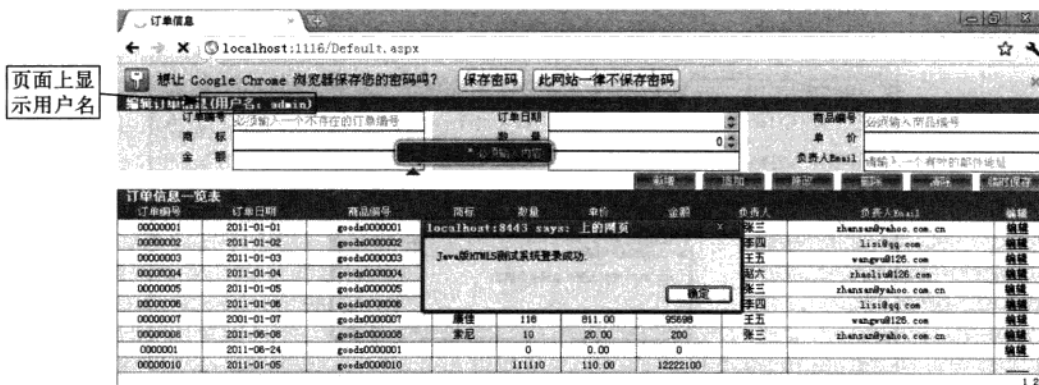


图 7-6 页面上提示成功登录 Java 版本的测试系统

在 ASP.NET 版本的测试系统中，登录成功后所显示页面的 URL 地址为“http://localhost:1116/Default.aspx”，如果用户不是在首页上输入并提交用户名与密码，而是直接访问该页面地址，则页面将被重定向到登录页面。

### 3. 用户身份统一认证系统的页面显示效果

最后来看一下本案例用户身份统一认证系统中所使用的登录页面在浏览器中的显示效果，如图 7-7 所示。

单击登录按钮后系统将用户名与密码发送到 Java 版本的测试系统与 ASP.NET 版本的测试系统中，这两个系统在接收到用户名与密码后，将自动实现用户登录功能，登录成功后弹出提示信息，提示用户登录成功，如图 7-8 与图 7-9 所示。

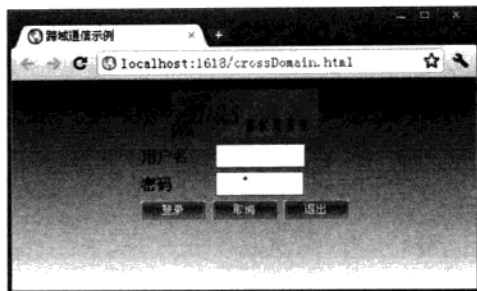


图 7-7 用户身份统一认证系统中所使用的登录页面

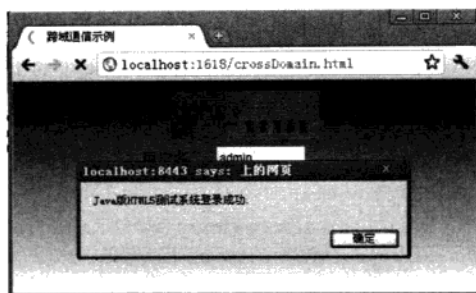


图 7-8 自动登录 Java 版本的测试系统成功后弹出提示信息

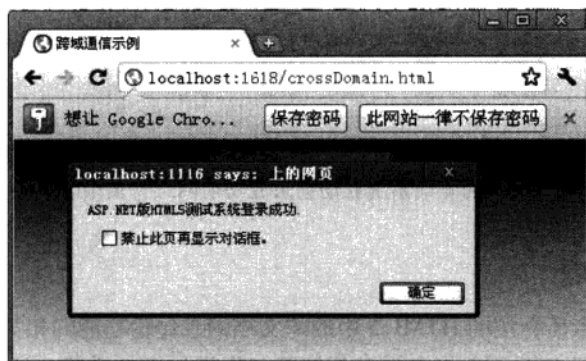


图 7-9 自动登录 ASP.NET 版本的测试系统成功后弹出提示信息

单击登录页面中的取消按钮，页面将被关闭，单击退出按钮，用户将同时退出 Java 版本的测试系统与 ASP.NET 版本的测试系统。

自动登录 Java 版本的测试系统成功后在浏览器中直接访问“http://localhost:8443/HTML5/”这个网站首页地址，浏览器中将打开 Java 版本的测试系统的欢迎页面，如图 7-10 所示。

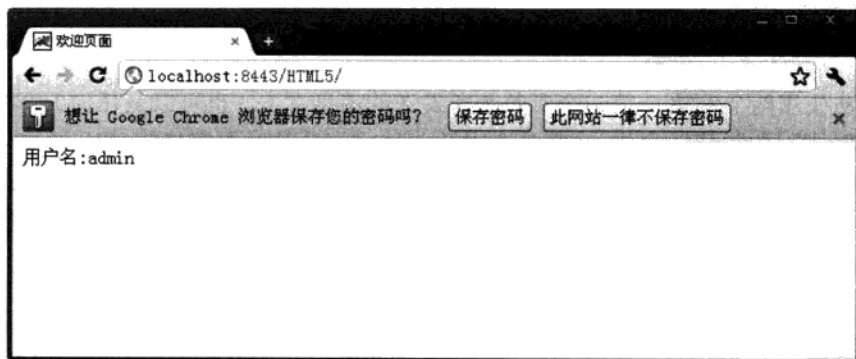


图 7-10 自动登录 Java 版本的测试系统成功后网站首页变为欢迎页面

ASP.NET 版本的测试系统自动登录成功后在浏览器中直接访问“http://localhost:1116/index.aspx”这个网站首页地址,页面将自动重定向到“http://localhost:1116/Default.aspx”这个 URL 地址,并打开本书第 2 章案例 4 所制作的页面,如图 7-11 所示。

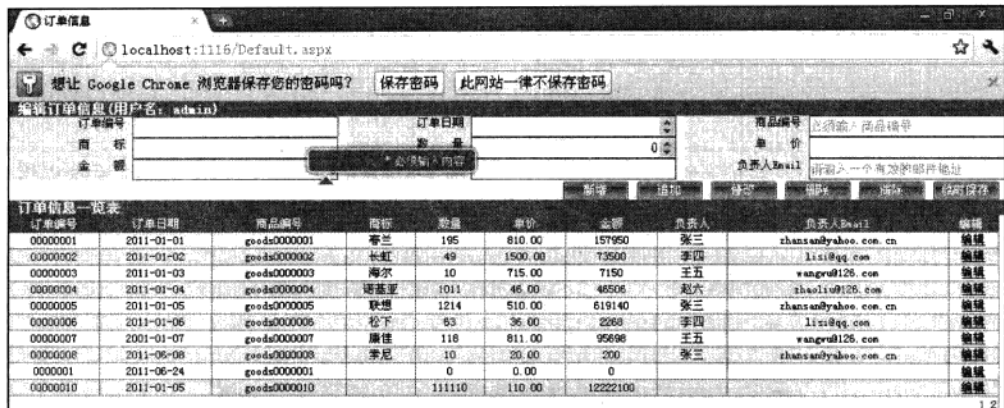


图 7-11 ASP.NET 版本的测试系统自动登录成功后网站首页地址变为订单输入页面

### 7.1.3 案例知识点

接下来看一下本章所介绍的 HTML 5 中的跨文档消息传输功能的相关知识及使用方法。

HTML 5 开始提供在网页文档之间互相接收与发送信息的功能。通过这个功能,只要获取网页所在窗口对象的实例,不仅可以实现同源(域+端口号)的 Web 网页之间互相通信,甚至可以实现跨域通信。

首先,要想接收从其他窗口发过来的消息,就必须对窗口对象的 message 事件进行监听,代码如下所示。

```
window.addEventListener("message", function() {...}, false);
```

使用 window 对象的 postMessage 方法向其他窗口发送消息,该方法的定义如下。

```
otherWindow.postMessage(message, targetOrigin);
```

该方法使用两个参数:第一个参数为所发送的消息文本,也可以是任何 JavaScript 对象(通过 JSON 将对象转换为文本),第二个参数为接收消息的对象窗口的 URL 地址(如 http://localhost:8080/).可以在 URL 地址字符串中使用通配符“\*”指定全部地址,建议使用准确的 URL 地址。otherWindow 为要发送窗口的对象的引用,可以通过 window.open 返回该对象,或通过为 window.frames 数组指定序号(index)或名字的方式来返回单个 frame 所属的窗口对象。

### 7.1.4 代码剖析

接下来详细分析本案例中的 Java 版本的对跨文档消息传输功能进行测试的 Web 系统、

ASP.NET 版本的对跨文档消息传输功能进行测试的 Web 系统、用户身份统一认证的 Web 系统的完整代码，以及这三个系统之间的数据传递过程，使读者完整地了解如何利用 HTML 5 中的跨文档消息传输功能来实现多个系统之间的单点登录功能。

### 1. Java 版本的 Web 系统

首先来看一下通过 Java 语言中的 ssh (struts2+spring2+hibernate) 框架开发出来的用来测试 HTML 5 中跨文档消息传输功能的 Web 系统的完整代码，在介绍每个页面文件及相关类文件中的代码之前，先来看一下本系统中几个必须使用的配置文件中的代码。

#### □ 配置文件

web.xml 配置文件中的代码如代码清单 7-1 所示。

代码清单 7-1 web.xml 配置文件中的代码

---

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_9" version="2.4"
    xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
    <listener>
        <listener-class>
            org.springframework.web.context.ContextLoaderListener
        </listener-class>
    </listener>
    <listener>
        <listener-class>
            org.springframework.web.context.request.RequestContextListener
        </listener-class>
    </listener>
    <filter>
        <filter-name>struts-cleanup</filter-name>
        <filter-class>
            org.apache.struts2.dispatcher.ActionContextCleanUp
        </filter-class>
    </filter>
    <filter-mapping>
        <filter-name>struts-cleanup</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
    <filter>
        <filter-name>struts2</filter-name>
        <filter-class>
            org.apache.struts2.dispatcher.FilterDispatcher
        </filter-class>
    </filter>
    <filter-mapping>
        <filter-name>struts2</filter-name>
```

```

        <url-pattern>*/</url-pattern>
    </filter-mapping>
    <servlet>
        <servlet-name>JspSupportServlet</servlet-name>
        <servlet-class>
            org.apache.struts2.views.JspSupportServlet
        </servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <welcome-file-list>
        <welcome-file>indexAction.action</welcome-file>
    </welcome-file-list>
</web-app>

```

注意，在 welcome-file-list 中，指定 welcome-file 属性的属性值为 indexAction.action，这样当用户在浏览器中输入整个 Web 系统的访问地址 “http://localhost:8443/HTML5/ ” 时，该 Web 系统的后台程序将会调用 indexAction.action 文件，在该文件中指定程序判断用户是否已经登录，如果尚未登录则打开用户登录页面，如果已经登录则打开欢迎页面。

struts.xml 配置文件中的代码如代码清单 7-2 所示。

代码清单 7-2 struts.xml 配置文件中的代码

```

<?xml version="1.0"?>
<!DOCTYPE struts PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
"http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <constant name="struts.enable.DynamicMethodInvocation"
        value="false" />
    <constant name="struts.devMode" value="true" />
    <constant name="struts.i18n.encoding" value="UTF-8" />
    <constant name="struts.custom.i18n.resources" value="messenger" />
    <constant name="struts.ui.theme" value="simple"/>
    <package name="Actions" extends="struts-default">
        <action name="login" class="Actions.LoginAction">
            <result>/index.jsp</result>
            <result type="redirectAction" name="success1">
                <param name="actionName">/welcome</param>
                <param name="crossDomainFlag">${crossDomainFlag}</param>
            </result>
        </action>
        <action name="welcome" class="Actions.welcomeAction">
            <result>/welcome.jsp</result>
        </action>
        <action name="logoff" class="Actions.LogoffAction">
            <result>/index.jsp</result>
        </action>
        <action name="indexAction" class="Actions.indexAction">

```

```

        <result>/index.jsp</result>
        <result name="success1">/welcome.jsp</result>
    </action>
</package>
</struts>

```

在本案例中，struts.xml 配置文件的作用主要是指定页面与页面之间的关系，指定当一个页面被提交后浏览器中打开哪一个页面，因此下文将结合页面及其后台程序来对 struts.xml 配置文件中的代码进行详细分析。

applicationContext.xml 配置文件中的代码如代码清单 7-3 所示。

代码清单 7-3 applicationContext.xml 配置文件中的代码

```

<?xml version="1.0"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
"http://www.springframework.org/dtd/spring-beans-2.0.dtd">
<beans default-autowire="autodetect">
    <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"
        destroy-method="close">
        <property name="driverClassName"
            value="com.microsoft.sqlserver.jdbc.SQLServerDriver" />
        <property name="url"
            value="jdbc:sqlserver://localhost:1433;databaseName=html5test" />
        <property name="username" value="aaa" />
        <property name="password" value="aaaaaaa"/>
    </bean>
    <!-- SessionFactory -->
    <bean id="sessionFactory"
        class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
        <property name="dataSource">
            <ref bean="dataSource"/>
        </property>
        <property name="configLocation">
            <value>classpath:hibernate.cfg.xml</value>
        </property>
        <property name="hibernateProperties">
            <value>
                hibernate.dialect=org.hibernate.dialect.SQLServerDialect
            </value>
        </property>
    </bean>
    <!-- DAO -->
    <bean id="userDao" class="data.impl.userImpl">
        <property name="sessionFactory">
            <ref bean="sessionFactory"/>
        </property>
    </bean>
    <!-- Services -->
    <bean id="userService" class="Spring.impl.userService">

```



```

        <property name="userDao">
            <ref bean="userDao"/>
        </property>
    </bean>
</beans>

```

这个配置文件整合了 Spring 层和 Hibernate 层的配置文件，将 BasicDataSource 注入到 hibernate sessionFactory 中，以得到数据库连接。同时配置了要使用的数据库的各种属性，spring 层中使用的各种业务层的类与接口，Hibernate 数据库层中使用的各种类与接口，以及它们之间的相互调用关系。稍后将结合页面对其中几个关键的业务层类与接口、数据库层类与接口进行详细分析。

hibernate.cfg.xml 配置文件中的代码如代码清单 7-4 所示。

代码清单 7-4 hibernate.cfg.xml 配置文件中的代码

```

<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <!-- Echo all executed SQL to stdout -->
        <property name="show_sql">true</property>
        <mapping resource="data/model/users.hbm.xml"/>
    </session-factory>
</hibernate-configuration>

```

在 hibernate.cfg.xml 配置文件中，只映射了一个数据表 users，在该表中存放了本系统中所有用户的用户名与密码。

users.hbm.xml 配置文件中的代码如代码清单 7-5 所示。

代码清单 7-5 users.hbm.xml 配置文件中的代码

```

<?xml version="1.0" encoding="GBK"?>
<!DOCTYPE hibernate-mapping
PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="data.model.users" table="users">
        <id name="id">
            <column name="id"/>
            <generator class="native" />
        </id>
        <property name="userName">
            <column name="userName" length="50" />
        </property>
        <property name="userPass">

```

```

        <column name="userPass" length="50" />
    </property>
</class>
</hibernate-mapping>

```

---

在该配置文件中，将 users 表中的 userName 与 userPass 字段映射为 users 类的 userName 与 userPass 属性。

#### ❑ indexAction 类文件

indexAction 类文件中的代码如代码清单 7-6 所示。

代码清单 7-6 indexAction 类文件中的代码

---

```

package Actions;
import com.opensymphony.xwork2.ActionContext;
import com.opensymphony.xwork2.ActionSupport;
public class indexAction extends ActionSupport{
    private String userName;
    private String userPass;
    public String getUserName() {
        return userName;
    }
    public void setUserName(String userName) {
        this.userName = userName;
    }
    public String getUserPass() {
        return userPass;
    }
    public void setUserPass(String userPass) {
        this.userPass = userPass;
    }
    public String execute()
    {
        ActionContext ctx=ActionContext.getContext();
        if(ctx.getSession().get("user")==null)
            return SUCCESS;
        else
        {
            this.userName =ctx.getSession().get("user").toString();
            this.userPass=ctx.getSession().get("pass").toString();
            return "success1";
        }
    }
}

```

---

当用户在浏览器中输入整个 Web 系统的访问地址 “http://localhost:8443/HTML5/ ” 时将调用该文件中的 execute 函数，在该函数中判断用户是否已经登录，如果用户尚未登录则 session 为空，因此 session 中的 user 值（表示登录用户的用户名）为 null，这时函数返回

SUCCESS，否则将 userName 属性的属性值设定为 session 中的 user 值，将 userPass 属性的属性值设定为 session 中的 pass 值（表示登录用户的用户密码），当用户在用户身份统一认证系统中登录或在 ASP.NET 版本的测试系统中登录成功后，会将用户名与用户密码发送到本系统中，本系统在接收到用户名与用户密码后会自动登录，自动登录成功后用户在浏览器中输入整个 Web 系统的访问地址“http://localhost:8443/HTML5/ ”时，session 中的 user 值与 pass 值为登录用户的用户名与用户密码。

在 struts.xml 配置文件中指定，当用户在浏览器中输入整个 Web 系统的访问地址“http://localhost:8443/HTML5/ ”时，如果用户未登录则打开用户登录页面，如果用户已经登录则打开欢迎页面，代码如下所示。

```
<action name="indexAction" class="Actions.indexAction">
    <result>/index.jsp</result>
    <result name="success1">/welcome.jsp</result>
</action>
```

□ 用户登录页面

用户登录页面中使用的控件信息如表 7-1 所示。

表 7-1 用户登录页面中使用的控件信息

控件名称	使用元素	显示文字	说明
log 图片	img		
用户名文字	页面文字	用户名	
用户名文本框	s:textfield		最大输入位数为 16
密码文字	页面文字	密码	
密码文本框	s:password		最大输入位数为 16
登录按钮	input type="button"	登录	单击登录按钮后提交表单，进行登录操作
取消按钮	input type="button"	取消	单击取消按钮后关闭页面
跨域通信标志隐藏文本框	s:hidden		在接收到其他页面传过来的用户名与密码后将文本框的内容设为“true”

用户登录页面的 HTML 页面代码与样式代码如代码清单 7-7 所示。

代码清单 7-7 用户登录页面的 HTML 页面代码与样式代码

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<%@ taglib prefix="s" uri="/struts-tags"%>
<html>
<head>
    <title> 登录 </title>
</head>
<style type="text/css">
input[type="text"],input[type="password"]{
```

```

        width: 90px;
        height: 20px;
        font-size: 12px;
    }
    input [type="button"] {
        font-size: 12px;
        width: 68px;
        height: 20px;
        cursor: hand;
        border: none;
        font-family: 宋体;
        background-color: White;
        background-image: url(images/but_bg.gif);
        color: white;
    }
    body {
        background-image: url(images/backbar.JPG);
    }
    img {
        width: 157px;
        height: 52px;
    }
</style>
<script language="javascript">
... 脚本代码稍后介绍 ...
</script>
<body>
<s:form id="LoginForm" name="LoginForm" action="login">
<center>
<table border="0">
    <tr>
        <td colspan="2">
            <div align="center"></div>
        </td>
    </tr>
    <tr>
        <td><font size="3"><b>用户名</b></font></td>
        <td>
            <s:textfield id="userName" maxlength="16" name="userName"/>
        </td>
    </tr>
    <tr>
        <td><font size="3"><b>密码</b></font></td>
        <td>
            <s:password id="userPass" maxlength="16" name="userPass">
            </s:password>
        </td>
    </tr>
    <tr>
        <td colspan="2" align="center">

```

```

        <input type="button" value=" 登录 " onclick="Login_Btn_Click()"/>
        <input type="button" value=" 取消 " onClick="Cancel_Btn_Click()"/>
    </td>
</tr>
</table>
</center>
<s:hidden id="crossDomainFlag" name="crossDomainFlag"></s:hidden>
</s:form>
</body>
</html>

```

用户登录页面的 JavaScript 脚本代码如代码清单 7-8 所示。

代码清单 7-8 用户登录页面的 JavaScript 脚本代码

```

<script language="javascript">
window.addEventListener("message", function(ev) {
    if (ev.origin != "http://localhost:1618"&&
        ev.origin != "http://localhost:1116") {
        return;
    }
    var data=JSON.parse(ev.data);
    document.getElementById("userName").value=data.userName;
    document.getElementById("userPass").value=data.userPass;
    document.getElementById("crossDomainFlag").value="true";
    document.getElementById("LoginForm").submit();
}, false);
moveTo((screen.width-400)/2, (screen.height-300)/2);
resizeTo(400,340);
var t;
t="<s:property value='tip' />";
if(t!="")
<s:if test="crossDomainFlag!=null&&!crossDomainFlag.equals('')">
alert("Java 版 HTML5 测试系统登录失败 \r\n 错误: "+t);
</s:if>
<s:else>
alert(t);
</s:else>
if(document.getElementById("crossDomainFlag").value=="logoff")
    alert("Java 版 HTML5 测试系统成功退出。");
function Login_Btn_Click() {
    if(document.getElementById("userName").value=="")
        alert(" 必须输入用户名。");
    else if(document.getElementById("userPass").value=="")
        alert(" 必须输入密码。");
    else
        document.getElementById("LoginForm").submit();
}
function Cancel_Btn_Click() {
    window.close();
}

```

```
}
</script>
```

在 JavaScript 脚本代码的开头，定义本页面在接收到其他系统传过来的数据后执行的代码。如果发送数据的域名及端口号不为“http://localhost:1618”（用户身份统一认证系统的域名及端口号）或“http://localhost:1116”（ASP.NET 版本测试系统的域名及端口号），则不执行任何动作，这样可以确保本系统的安全性，不处理其他域名及端口号传过来的任何数据。在接收到用户身份统一认证系统或 ASP.NET 版本测试系统传送过来的数据后，使用 JSON 对象的 parse 方法将数据解析为用户名与密码，然后将页面上的用户名文本框中的内容设定为接收到的用户名，将密码文本框中的内容设定为接收到的密码，将隐藏控件中的值设定为“true”，表示接收到其他系统发送过来的用户名与密码，最后执行表单提交，进行自动登录。

在接下来的脚本代码中，将页面宽度设定为 400 个像素，高度设定为 340 个像素，并将页面移动到屏幕中央。随后，判断提交页面后用户登录是否成功，如果不成功则判断提交前隐藏控件中的值是否为空，如果为空则设定弹出文字为提交后服务器端脚本代码传过来的错误提示信息，否则设定弹出文字为“Java 版 HTML5 测试系统登录失败 \r\n 错误：”加上后台脚本代码传过来的错误提示信息，其中“\r\n”表示将弹出文字进行换行（当接收到其他系统发送过来的用户名与密码并执行自动登录时将弹出此提示文字）。

另外，如果本页面是用户在用户身份统一认证系统中单击退出按钮自动退出本系统后打开的页面，则服务器端脚本代码中会将页面上隐藏控件中的值设定为“logout”，这时弹出提示信息，文字为“Java 版 HTML5 测试系统成功退出。”。

接下来，定义两个函数，用户单击登录按钮时将调用 Login\_Btn\_Click 函数。在该函数中进行判断，如果用户未输入用户名，则弹出提示信息，文字为“必须输入用户名。”。如果用户未输入密码，则弹出提示信息，文字为“必须输入密码。”。如果用户输入了用户名与密码，则执行表单提交，执行用户登录。用户单击取消按钮时将调用 Cancel\_Btn\_Click 函数将页面关闭。

#### ❑ LoginAction 类文件

用户在登录页面中输入用户名与密码后单击登录按钮进行登录，提交表单后将调用 LoginAction 类文件，当系统接收到其他系统发送的用户名与密码后执行自动登录时也将调用该文件，该文件中的代码如代码清单 7-9 所示。

代码清单 7-9 LoginAction 类文件中的代码

```
package Actions;
import com.opensymphony.xwork2.ActionContext;
import com.opensymphony.xwork2.ActionSupport;
import Spring.IFace.userServiceIFace;
import common.AbstractAction;
import common.CommonServiceIFace;
import javax.servlet.http.HttpServletRequest;
import org.apache.struts2.ServletActionContext;
```

```

public class LoginAction extends ActionSupport{
    private static final long serialVersionUID = 1L;
    private userServiceIFace userService;
    private CommonServiceIFace commonService;
    private String userName;
    private String userPass;
    private String tip;
    private String crossDomainFlag="";
    public userServiceIFace getUserService()
    {
        return this.userService;
    }
    public void setUserService(userServiceIFace userService)
    {
        this.userService=userService;
    }
    public String getCrossDomainFlag() {
        return crossDomainFlag;
    }
    public void setCrossDomainFlag(String crossDomainFlag) {
        this.crossDomainFlag = crossDomainFlag;
    }
    public String getTip() {
        return tip;
    }
    public void setTip(String tip) {
        this.tip = tip;
    }
    public String getUsername()
    {
        return this.userName;
    }
    public void setUsername(String userName)
    {
        this.userName=userName;
    }
    public String getUserPass()
    {
        return this.userPass;
    }
    public void setUserPass(String userPass)
    {
        this.userPass=userPass;
    }
    public String execute()
    {
        String userName,userPass;
        userName= this.userName.trim().replace("'", "");
        userPass= this.userPass.trim().replace("'", "");
        int i=this.userService.QueryUser(userName,userPass);
    }
}

```

```

        ActionContext ctx=ActionContext.getContext();
        switch(i)
        {
            case -1:
                tip=this.userService.getErrString();
                break;
            case 1:
                tip=" 输入的用户名不正确。";
                break;
            case 2:
                tip=" 输入的密码不正确。";
                break;
            case 3:
                ctx.getSession().put("user",this.userName);
                ctx.getSession().put("pass",this.userPass);
                if(this.crossDomainFlag!=null&&
                    (!this.crossDomainFlag.equals("")))
                {
                    HttpServletRequest request=
                        (HttpServletRequest)
                            ctx.get(ServletActionContext.HTTP_REQUEST);
                    request.setAttribute("crossDomainFlag",Boolean.TRUE);
                }
                return "success1";
        }
        return SUCCESS;
    }
}

```

用户在登录页面中输入用户名与密码后单击登录按钮进行登录，提交表单后将调用 LoginAction 类文件中的 execute 函数，当系统接收到其他系统中发送的用户名与密码后执行自动登录时也将调用该文件中的 execute 函数。该函数首先调用 userService 类中的 QueryUser 函数到数据库中查询提交的用户名与密码是否存在，如果查询时发生系统错误，则调用 userService 类中的 getErrString 函数获取系统错误信息，同时将其设定为服务器端返回的错误信息；如果提交的用户名错误，则将服务器端返回的错误信息设定为“输入的用户名不正确。”；如果提交的密码错误，则将服务器端返回的错误信息设定为“输入的密码不正确。”；否则将 session 中的 user 值设定为提交的用户名，将 session 中的 pass 值设定为提交的用户密码，如果页面上隐藏控件中的内容不为空白（当系统接收到其他系统发送过来的用户名与密码时该隐藏控件中的内容被设定为“true”），则设定本页面传给下一个页面或 action 的参数 crossDomainFlag 的值为 true。用户登录成功后 execute 函数返回 success1，否则返回 SUCCESS。在 struts.xml 配置文件中指定用户登录成功后调用 welcomeAction 类文件，用户登录失败时返回用户登录页面，代码如下。

```

<action name="login" class="Actions.LoginAction">
    <result>/index.jsp</result>

```



```

<result type="redirectAction" name="success1">
  <param name="actionName">/welcome</param>
  <param name="crossDomainFlag">${crossDomainFlag}</param>
</result>
</action>

```

□ userServiceIFace 接口文件与 userService 类文件

userServiceIFace 接口文件中的代码如代码清单 7-10 所示。

代码清单 7-10 userServiceIFace 接口文件中的代码

---

```

package Spring.IFace;
public interface userServiceIFace {
    public int QueryUser(String userName,String userPass);
    public String getErrString();
}

```

---

userService 类文件继承 userServiceIFace 接口文件，当用户在登录页面中单击登录按钮或系统接收到其他系统中发送的用户名与密码后执行自动登录时将调用 LoginAction 类文件中的 execute 函数，该函数调用 userService 类文件中的 QueryUser 函数在数据库中查询用户名与密码是否存在。

userService 类文件中的代码如代码清单 7-11 所示。

代码清单 7-11 userService 类文件中的代码

---

```

package Spring.impl;
import data.iFace.userFace;
import data.model.users;
import Spring.IFace.*;
public class userService implements userServiceIFace{
    private userFace userDao;
    private String errString;
    public UserServive(){}
    public int QueryUser(String userName,String userPass){
        int x;
        x=userDao.QueryUser(userName, userPass);
        if(x== -1)
            errString=userDao.getErrString();
        return x;
    }
    public userFace getUserDao() {
        return this.userDao;
    }
    public void setuserDao(userFace userDao) {
        this.userDao = userDao;
    }
    public String getErrString()
    {

```

---

```

        return this.errString;
    }
}

```

QueryUser 函数接收两个参数，userName 表示用户名，userPass 表示密码，在函数内调用 Hibernate 数据库层的 userImpl 类文件中的 QueryUser 函数在数据库中查询用户名与密码是否存在，如果查询时发生系统错误，则该函数返回 -1，这时调用 userImpl 类文件中的 getErrString 函数获取系统错误信息并将其赋值给本类中的私有变量 errString，这样 LoginAction 类文件中的 execute 函数就能够通过调用本类中的 getErrString 来获取系统错误信息了。如果查询时没有发生系统错误，则直接返回 userImpl 类文件中的 QueryUser 函数的返回值。

在 applicationContext.xml 配置文件中指定 userService 类文件调用 Hibernate 数据库层的 userImpl 类文件，代码如下所示。

```

<!-- DAO -->
<bean id="userDao" class="data.impl.userImpl">
    <property name="sessionFactory">
        <ref bean="sessionFactory"/>
    </property>
    <property name="messages"><ref bean="messageSource"/></property>
</bean>
<!-- Services -->
<bean id="userService" class="Spring.impl.userService">
    <property name="userDao">
        <ref bean="userDao"/>
    </property>
</bean>

```

#### ❑ userFace 接口文件、userService 类文件与 users 类文件

在 Hibernate 数据库层，users 数据表被映射为 users 类文件，该文件内的代码如代码清单 7-12 所示。

代码清单 7-12 users 类文件中的代码

```

package data.model;
public class users {
    private int id;
    private String userName;
    private String userPass;
    public int getId()
    {
        return this.id;
    }
    public void setId(int id)
    {
        this.id=id;
    }
}

```

```

    }
    public String getUserName(){return userName;}
    public void setUsername(String userName){this.userName=userName;}
    public String getUserPass(){return userPass;}
    public void setUserPass(String userPass){this.userPass=userPass;}
}

```

---

userFace 接口文件中的代码如代码清单 7-13 所示。

代码清单 7-13 userFace 接口文件中的代码

```

package data.iFace;
public interface userFace {
    public int QueryUser(String userName,String userPass);
    public String getErrString();
}

```

---

userImpl 类文件继承 userFace 接口文件, userService 类文件中的 QueryUser 函数调用 userImpl 类文件中的 QueryUser 函数在数据库中查询用户名与密码是否存在。userImpl 文件中的代码如代码清单 7-14 所示。

代码清单 7-14 userImpl 类文件中的代码

```

package data.impl;
import java.util.Iterator;
import java.util.List;
import org.springframework.context.MessageSource;
import org.springframework.orm.hibernate3.support.HibernateDaoSupport;
import data.iFace.*;
import org.hibernate.exception.*;
public class userImpl extends HibernateDaoSupport implements userFace{
    private String errString;
    public int QueryUser(String userName,String userPass)
    {
        try
        {
            String sql="select userPass FROM users where
            userName='"+userName+"'";
            String tmpPass=null;
            List ll = (List) this.getHibernateTemplate().find(sql);
            Iterator itr = ll.iterator();
            if (itr.hasNext()) {
                Object to = itr.next();
                tmpPass = to.toString();
            }
            if (tmpPass==null)
                return 1;
            else if (!tmpPass.equals(userPass))
                return 2;

```

```

        else
            return 3;
    }
    catch(Exception ex)
    {
        this.errString=ex.getMessage();
        return -1;
    }
}
public String getErrString()
{
    return this.errString;
}
}

```

QueryUser 函数接收两个参数，userName 表示用户名，userPass 表示密码，该函数在数据库中查询用户名与密码是否存在，如果查询时发生系统错误则返回 -1 并将系统错误信息赋值给本类中的私有变量 errString，这样 userService 类文件中的 QueryUser 函数就能够通过调用 userImpl 类中的 getErrString 来获取系统错误信息。如果用户名在数据表中不存在，则返回 1；如果用户的密码与传入的密码不相等，则返回 2；否则返回 3，表示用户名与用户密码正确。

#### ❑ welcomeAction 类文件

当用户在登录页面中输入用户名与密码并单击登录按钮进行表单提交后，或者本系统接收到用户身份统一认证系统或 ASP.NET 版本的测试系统发送的用户名与密码并自动提交后，系统将调用 indexAction 类文件并在数据库中查询用户名与密码是否正确，如果正确则调用 welcomeAction 类文件中的 execute 函数，该类文件中的代码如代码清单 7-15 所示。

代码清单 7-15 welcomeAction 类文件中的代码

```

package Actions;
import com.opensymphony.xwork2.ActionContext;
import com.opensymphony.xwork2.ActionSupport;
public class welcomeAction extends ActionSupport{
    private String userName;
    private String userPass;
    private Boolean crossDomainFlag;
    public String getUser_name() {
        return userName;
    }
    public void setUser_name(String userName) {
        this.userName = userName;
    }
    public String getUserPass() {
        return userPass;
    }
    public void setUserPass(String userPass) {

```

```

        this.userPass = userPass;
    }
    public Boolean getCrossDomainFlag() {
        return crossDomainFlag;
    }
    public void setCrossDomainFlag(Boolean crossDomainFlag) {
        this.crossDomainFlag = crossDomainFlag;
    }
    public String execute()
    {
        ActionContext ctx=ActionContext.getContext();
        if(ctx.getSession().get("user")!=null)
        {
            this.userName =ctx.getSession().get("user").toString();
            this.userPass=ctx.getSession().get("pass").toString();
        }
        return SUCCESS;
    }
}

```

在 execute 函数中，首先判断用户是否成功登录，如果成功登录，则将 session 中的 user 值（代表用户名）赋值给本类中的私有变量 userName（用于在欢迎页面上显示用户名以及将该用户名发送给 ASP.NET 版本的测试系统），将 session 中的 pass 值（代表密码）赋值给本类中的私有变量 userPass（用于将该密码发送给 ASP.NET 版本的测试系统），然后返回 SUCCESS，在 struts.xml 配置文件中指定调用 welcomeAction 类文件中的 execute 函数后在浏览器中打开欢迎页面，代码如下所示。

```

<action name="welcome" class="Actions.welcomeAction">
<result>/welcome.jsp</result>
</action>

```

#### □ 欢迎页面

接下来看一下欢迎页面中的完整代码，如代码清单 7-16 所示。

代码清单 7-16 欢迎页面中的完整代码

```

<%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
<!DOCTYPE html>
<%@ taglib prefix="s" uri="/struts-tags"%>
<html>
<head>
<title> 欢迎页面 </title>
<style type="text/css">
iframe{
    display:none;
}
</style>
<script language="javascript">

```

```

function iframe_load()
{
    <s:if test="crossDomainFlag==true">
        alert("Java 版 HTML5 测试系统登录成功。");
    </s:if>
    <s:elseif test="userName!=null&&!userName.equals('')">
        var data = new Object();
        data.userName = "<s:property value='userName' />";
        data.userPass = "<s:property value='userPass' />";
        var iframe = window.frames[0];
        iframe.postMessage(JSON.stringify(data), "http://localhost:1116");
    </s:elseif>
}
</script>
</head>
<s:if test="userName!=null&&!userName.equals('')">
<body>
    用户名:<s:property value='userName' />
    <iframe src="http://localhost:1116" onload="iframe_load()"></iframe>
</body>
</s:if>
</html>

```

欢迎页面（welcome.jsp 文件）中的 HTML 页面代码比较简单，只是用户登录成功后在页面上显示一下用户名，同时放置一个隐藏的 iframe，将 iframe 中的页面设定为 ASP.NET 版本的测试系统的用户登录页面（即该系统的首页），用于当用户在本系统的用户登录页面中输入用户名与密码并登录成功后将用户名与密码发送给该 iframe 的远程页面。另外，如果非法用户在浏览器中输入 welcome.action 类文件的 URL 地址或本页面的 URL 地址，则 session 为空，因此服务器端传过来的 userName 变量值必然为空，这时设置页面上不显示任何内容。

欢迎页面中的 JavaScript 脚本代码部分比较简单，只有一个 iframe\_load 函数，当页面上 iframe 中的页面装载完成后将调用该函数。在该函数中进行判断，如果本页面为当用户在其他系统（用户身份统一认证系统或在 ASP.NET 版本的测试系统）中登录成功后，该系统发送用户名与密码给本系统，在本系统中执行自动登录后打开的页面，则由于在用户登录页面中已设定 crossDomainFlag 隐藏控件的值为 true，在 indexAction 类文件中已设定变量 crossDomainFlag 值为 true，并且通过 URL 参数传递给 welcomeAction 类文件，因此在 JavaScript 脚本代码中可以查询到服务器端的 crossDomainFlag 变量的变量值为 true，这时弹出提示信息框，文字为“Java 版 HTML5 测试系统登录成功。”

如果本页面为用户直接在本系统的用户登录页面中输入用户名与密码后单击登录按钮执行登录后打开的页面，则将用户名与密码发送给 ASP.NET 版本的测试系统（URL 地址为 http://localhost:1116），通知该系统执行自动登录。

#### □ 用户退出登录页面

在用户身份统一认证系统中单击退出按钮后，会向本系统的用户退出登录页面发送消

息, 通知本系统将该用户注销。用户退出登录页面中的代码如代码清单 7-17 所示。

代码清单 7-17 用户退出登录页面中的代码

---

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<%@ taglib prefix="s" uri="/struts-tags"%>
<html>
<head>
<title>退出</title>
</head>
<script language="javascript">
window.addEventListener("message", function(ev) {
    if (ev.origin != "http://localhost:1618") {
        return;
    }
    var data=ev.data;
    if(data=="logout")
        document.getElementById("form").submit();
}, false);
</script>
<body>
<s:form id="form" action="logout">
</s:form>
</body>
</html>
```

---

本页面中的 HTML 页面代码非常简单, 仅提供一个空白表单, 然后在脚本代码中指定, 如果接收其他域或其他端口发送的数据, 全都不做处理。只有在接收到用户身份统一认证系统发送的数据时才将表单提交, 在服务器端将用户注销。

#### □ LogoutAction 类文件

用户退出登录页面接收到用户身份统一认证系统中发送过来的数据后将页面中的表单提交, 之后调用 LogoutAction 类文件中的 execute 函数, 该文件中的代码如代码清单 7-18 所示。

代码清单 7-18 LogoutAction 类文件中的代码

---

```
package Actions;
import com.opensymphony.xwork2.ActionContext;
import com.opensymphony.xwork2.ActionSupport;
public class LogoutAction extends ActionSupport{
    private static final long serialVersionUID = 1L;
    private String crossDomainFlag;
    public String getCrossDomainFlag() {
        return crossDomainFlag;
    }
    public void setCrossDomainFlag(String crossDomainFlag) {
        this.crossDomainFlag = crossDomainFlag;
    }
}
```

---

```
    }  
    public String execute()  
    {  
        ActionContext ctx=ActionContext.getContext();  
        ctx.getSession().clear();  
        this.crossDomainFlag ="logoff";  
        return SUCCESS;  
    }  
}
```

在 execute 函数中将用户的 session 清空，将服务器端私有变量 crossDomainFlag 的值设为“logoff”，然后返回 SUCCESS。在 struts.xml 配置文件中指定接下来在浏览器中打开用户登录页面，代码如下所示。

```
<action name="logoff" class="Actions.LogoffAction">  
<result>/index.jsp</result>  
</action>
```

由于服务器端私有变量 crossDomainFlag 的值被设为“logoff”，因此在用户登录页面中弹出提示信息，文字为“Java 版 HTML5 测试系统成功退出。”。

2. ASP.NET 版本的 Web 系统

接下来看一下本案例中使用 ASP.NET 制作的用来测试跨文档消息传输功能的 Web 系统的完整代码。

□ 用户登录页面

用户登录页面中使用的控件信息如表 7-2 所示。

表 7-2 用户登录页面中使用的控件信息

控件名称	使用元素	显示文字	说明
log 图片	img		
用户名文字	页面文字	用户名	
用户名文本框	asp:TextBox		最大输入位数为 16
密码文字	页面文字	密码	
密码文本框	asp:TextBox TextMode="Password"		最大输入位数为 16
登录按钮	asp:Button	登录	单击登录按钮后提交表单，进行登录操作
取消按钮	input type="button"	取消	单击取消按钮后关闭页面
跨域通信标志隐藏文本框	asp:HiddenField		在接收到其他页面传过来的用户名与密码后文本框的内容被设为“true”

用户登录页面的 HTML 页面代码与样式代码如代码清单 7-19 所示。



代码清单 7-19 用户登录页面的 HTML 页面代码与样式代码

```

<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="index.aspx.cs"
Inherits="HTML5TEST.index" %>
<!DOCTYPE html>
<html>
<head runat="server">
<title> 登录 </title>
<style type="text/css">
input [type="text"],input [type="password"] {
    width: 90px;
    height: 20px;
    font-size:12px;
}
input [type="submit"],input [type="button"] {
    font-size: 12px;
    width: 68px;
    height: 20px;
    cursor: hand;
    border:none;
    font-family: 宋体 ;
    background-color:White;
    background-image: url(images/but_bg.gif);
    color: white;
}
body{
    background-image:url(images/backbar.JPG);
}
img{
    width:157px;
    height:52px;
}
</style>
<script language="javascript">
... 脚本代码稍后介绍 ...
</script>
</head>
<body>
<form id="form1" runat="server" >
<center>
<table border="0">
    <tr>
        <td colspan="2">
            <div align="center">
                
            </div>
        </td>
    </tr>
    <tr>
        <td><font size="3"><b>用户名 </b></font></td>

```

```

        <td>
            <asp:TextBox id="userName" maxlength="16" name="userName"
                runat="server" ClientIDMode="Static"/>
        </td>
    </tr>
    <tr>
        <td><font size="3"><b>密码</b></font></td>
        <td>
            <asp:TextBox id="userPass" maxlength="16" name="userPass"
                runat="server" TextMode="Password" ClientIDMode="Static"/>
        </td>
    </tr>
    <tr>
        <td colspan="2" align="center">
            <asp:Button name="Login_Btn" Text=" 登录 "
                OnClientClick="return Login_Btn_Click();" runat="server"
                ID="Login_Btn" onclick="Login_Btn_Click"/>
            <input type="button" value=" 取消 " onclick="Cancel_Btn_Click()"/>
        </td>
    </tr>
</table>
</center>
<asp:HiddenField ID="crossDomainFlag" ClientIDMode="Static"
runat="server"/>
</form>
</body>
</html>

```

用户登录页面的 JavaScript 脚本代码如代码清单 7-20 所示。

代码清单 7-20 用户登录页面的 JavaScript 脚本代码

```

<script language="javascript">
window.addEventListener("message", function (ev) {
    if (ev.origin != "http://localhost:1618" &&
        ev.origin != "http://localhost:8443") {
        return;
    }
    var data = JSON.parse(ev.data);
    document.getElementById("userName").value = data.userName;
    document.getElementById("userPass").value = data.userPass;
    document.getElementById("crossDomainFlag").value = "true";
    document.getElementById("Login_Btn").click();
}, false);
moveTo((screen.width - 400) / 2, (screen.height - 300) / 2);
resizeTo(400, 340);
function Login_Btn_Click() {
    if (document.getElementById("userName").value.trim() == "")
    {
        alert(" 必须输入用户名。");
    }
}

```

```

        return false;
    }
    else if (document.getElementById("userPass").value.trim() == "")
    {
        alert("必须输入密码。");
        return false;
    }
    return true;
}
function Cancel_Btn_Click() {
    window.close();
}
</script>

```

在此段脚本代码的开头，定义了本页面在接收到其他系统传过来的数据后执行的代码。如果发送数据的域名和端口号不为“http://localhost:1618”（用户身份统一认证系统的域名及端口号）或“http://localhost:8443”（Java 版本测试系统的域名及端口号），则不执行任何动作，这样可以确保本系统的安全，不处理其他域名及端口号传过来的任何数据。在接收到用户身份统一认证系统或 Java 版本测试系统传送过来的数据后，使用 JSON 对象的 parse 方法将数据解析为用户名与密码，然后将页面上的用户名文本框中的内容设定为接收到的用户名，将密码文本框中的内容设定为接收到的密码，将隐藏控件中的值设定为“true”，表示接收到其他系统发送过来的用户名与密码，随后自动单击页面上的登录按钮，进行自动登录。

在接下来的脚本代码中，将页面宽度设定为 400 个像素，高度设定为 340 个像素，并将页面移动到屏幕中央。

接下来，定义两个函数，当单击登录按钮时将调用 Login\_Btn\_Click 函数，在该函数中进行判断，如果未输入用户名，则弹出提示信息，文字为“必须输入用户名。”。如果未输入密码，则弹出提示信息，文字为“必须输入密码。”。如果输入了用户名与密码，则该函数返回 true，调用服务器端的 Login\_Btn\_Click 函数，进行用户登录。用户单击取消按钮时将调用 Cancel\_Btn\_Click 函数将页面关闭。

提交该页面后的服务器端代码如代码清单 7-21 所示。

代码清单 7-21 提交用户登录页面后的服务器端代码

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Data;
using System.Data.SqlClient;
namespace HTML5TEST
{
    public partial class index : System.Web.UI.Page

```

```

{
    private System.Data.SqlClient.SqlConnection SqlCon;
    private System.Data.SqlClient.SqlDataAdapter DataAdapter;
    private int myErrorNumber = 0;
    private string myErrorMessage = "";
    private bool SuccessFlag;
    string Constr;
    public string strErrMsg;
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!Page.IsPostBack)
        {
            if (Request.QueryString["crossDomainFlag"] != null &&
                Request.QueryString["crossDomainFlag"].ToString() ==
                "logout")
            {
                Page.ClientScript.RegisterStartupScript(
                    Page.GetType(),
                    "",
                    "<script language='JavaScript'>alert('ASP.NET 版 HTML5 测试系统成功退出');</script>"
                );
                if (Session["UserName"] != null)
                    Response.Redirect("Default.aspx");
            }
        }
        else
        {
            Constr = System.Configuration.ConfigurationManager.
                ConnectionStrings["dbConnection"].ToString();
            this.SqlCon = new System.Data.SqlClient.SqlConnection();
            this.SqlCon.ConnectionString = Constr;
            SuccessFlag = OpenDBConnection();
            if (SuccessFlag == false)
            {
                this.ShowError("数据库连接失败。");
            }
        }
    }
    protected void Login_Btn_Click(object sender, EventArgs e)
    {
        string SqlStr;
        DataSet ds;
        SqlStr="select userPass from users where
        userName='"+this.userName.Text.Trim().Replace("'", "''")+"'";
        ds = this.GetDataFromDB(SqlStr);
        if (ds == null)
        {
            this.ShowError("数据库操作失败。");
            return;
        }
        else if (ds.Tables[0].Rows.Count == 0)
    }

```

```

    {
        this.ShowError(" 输入的用户名不正确。");
        return;
    }
    else if (ds.Tables[0].Rows[0][0].ToString() !=
this.userPass.Text.Trim())
    {
        this.ShowError(" 输入的密码不正确。");
        return;
    }
    Session["UserName"] = this.userName.Text.Trim();
    Session["UserPass"] = this.userPass.Text.Trim();
    if (this.crossDomainFlag.Value=="true")
        Response.Redirect("Default.aspx?crossDomainFlag=" +
            this.crossDomainFlag.Value);
    else
        Response.Redirect("Default.aspx");
}
private bool OpenDBConnection()
{
    try
    {
        this.SqlCon.Open();
        return true;
    }
    catch (SqlException e)
    {
        this.myErrorNumber = e.Number;
        this.myErrorMessage = e.Message;
        return false;
    }
    catch (Exception e)
    {
        this.myErrorNumber = e.GetHashCode();
        this.myErrorMessage = e.Message;
        return false;
    }
}
public System.Data.DataSet GetDataFromDB(string SqlStr)
{
    try
    {
        DataSet ds= new DataSet();
        this.DataAdapter = new SqlDataAdapter(SqlStr, this.SqlCon);
        this.DataAdapter.Fill(ds);
        return ds;
    }
    catch (SqlException e)
    {
        this.myErrorNumber = e.Number;

```

```

        this.myErrorMessage = e.Message;
        return null;
    }
    catch (Exception e)
    {
        this.myErrorNumber = e.GetHashCode();
        this.myErrorMessage = e.Message;
        return null;
    }
}

public void ShowError(string myErrorMessage)
{
    if (!this.myErrorMessage.Equals(""))
        strErrMsg = this.myErrorNumber + ":" + this.myErrorMessage;
    else
        strErrMsg = myErrorMessage;
    if (this.crossDomainFlag.Value == "")
        Page.ClientScript.RegisterStartupScript(
            Page.GetType(),
            "",
            "<script language='JavaScript'>alert('" + strErrMsg +
            "');</script>"
        );
    else
        Page.ClientScript.RegisterStartupScript(
            Page.GetType(),
            "",
            "<script language='JavaScript'>alert('ASP.NET 版 HTML5 测试系统
            登录失败 ( 错误 : " + strErrMsg + " )');</script>"
        );
}
}
}

```

在装载页面的时候进行判断，如果页面是首次装载而不是页面上的表单被提交后的再次装载，则先判断访问页面的 URL 地址中是否存在 crossDomainFlag 参数，如果存在该参数并且参数值为“logoff”（当用户在用户身份统一认证系统中单击退出按钮时，该页面会发送消息给本系统中的退出页面，退出页面在接收消息后执行表单提交，将用户从本系统中注销，同时将页面重定向到用户登录页面，在 URL 地址中将 crossDomainFlag 参数的值设定为“logoff”），则弹出提示文字，内容为“ASP.NET 版 HTML5 测试系统成功退出”。

如果用户已经登录本系统，则 session 中的 UserName 已经被设定为用户名，用户再次访问本页面时，将本页面重定向到本书第 2 章案例 4 制作的订单信息输入页面。

如果本页面为在用户登录页面中单击登录按钮，或者系统接收到用户身份统一认证系统或 Java 版本的测试系统中发送过来的用户名与密码后自动单击登录按钮，执行表单的自动提交后打开的页面，则先连接数据库，连接失败时会弹出错误提示信息，文字为“数据库连接

失败。”。

当登录按钮被用户单击或被自动单击时会调用 Login\_Btn\_Click 函数，该函数首先在数据库中查询用户名与密码是否存在，如果查询失败则弹出系统错误信息；如果用户名在数据表中不存在，则弹出提示信息，文字为“输入的用户名不正确。”；如果密码与数据表中该用户的密码不一致，则弹出提示信息，文字为“输入的密码不正确。”；如果用户名与密码正确，则将 session 中的 UserName 设定为用户名，将 session 中的 UserPass 设定为密码，然后将页面重定向到订单信息输入页面，如果隐藏控件中的内容为“true”（系统在向用户身份统一认证系统或 Java 版本的测试系统中发送的用户名与密码时会将隐藏控件中的内容设定为“true”），则同时将重定向到订单信息输入页面的页面参数 crossDomainFlag 的值设定为“true”。

#### □ 订单信息输入页面

接下来看一下本案例对第 2 章案例 4 所制作的订单信息输入页面做了哪些必要的修改。

首先，在原页面的“编辑订单信息”文字旁边，添加了显示登录用户名的代码，如下所示。

编辑订单信息（用户名：<%=userName%>）

当用户在用户登录页面中输入用户名与密码后单击登录按钮进行登录时，为了能够将用户名与密码发送给 Java 版本的 Web 测试系统，在页面上添加隐藏的 iframe 元素，在该 iframe 元素中装载 Java 版本的 Web 测试系统的首页，代码如下所示。

```
<iframe src="http://localhost:8443/HTML5" onload="iframe_load()"></iframe>
```

在样式代码中，设定 iframe 为隐藏状态，代码如下所示。

```
iframe{
    display:none;
}
```

在 JavaScript 脚本代码中，添加页面上的 iframe 元素装载完成后调用的 iframe\_load，代码如下所示。

```
function iframe_load()
{
    <%if(crossDomainFlag)
    {%>
        alert("ASP.NET 版 HTML5 测试系统登录成功。");
    <%
    }
    else
    {
    %>
        var data = new Object();
        data.userName = "<%=userName%>";
        data.userPass = "<%=userPass%>";
        var iframe = window.frames[0];
```

```

        iframe.postMessage(JSON.stringify(data),
            "http://localhost:8443/HTML5");
    }
}
%>
}

```

在该函数中进行判断，如果本页面为当用户在用户身份统一认证系统中或在 Java 版本的测试系统中登录成功后发送用户名与密码给本系统并执行自动登录后打开的页面，则由于在用户登录页面中已设定 crossDomainFlag 隐藏控件的值为 true，并且通过 URL 参数传递给本页面，因此在 JavaScript 脚本代码中可以查询到服务器端的 crossDomainFlag 变量的变量值为 true，这时弹出提示信息框，文字为“ASP.NET 版 HTML5 测试系统登录成功。”

如果 URL 参数 crossDomainFlag 不存在，则表示本页面是用户在用户登录页面中输入用户名与密码，然后单击登录按钮执行表单提交后打开的页面，这时将用户输入的用户名与密码发送到 Java 版本的 Web 测试系统，通知该系统执行自动登录。

在服务器端代码中添加如下变量：

```

public String userName;// 用户名
public String userPass;// 用户密码
public bool crossDomainFlag;// 是否接收到其他系统发送过来的用户名与密码

```

在服务器端的 Page\_Load（打开页面）事件代码中添加如下代码：

```

if (Session["UserName"] != null)
{
    userName = Session["UserName"].ToString();
    userPass = Session["UserPass"].ToString();
}
else
    Response.Redirect("index.aspx");
if (Request.QueryString["crossDomainFlag"] != null &&
    Request.QueryString["crossDomainFlag"].ToString() == "true")
    crossDomainFlag = true;

```

这段代码表示如果用户已经成功登录，则将本页面中的 userName 与 userPass 变量值设定为 session 中的 userName 与 userPass 变量值，用于在页面上显示用户名，将用户名与密码发送给 Java 版本的 Web 测试系统（如果本页面为当用户在用户身份统一认证系统中或在 Java 版本的测试系统中登录成功后发送用户名与密码给本系统并执行自动登录后打开的页面）。由于非法用户直接访问本页面时 session 为空，因此 userName 为 null，这时将页面重定向到用户登录页面。如果 URL 地址栏中的参数 crossDomainFlag 的值为“true”，则将本页面中的公有变量 crossDomainFlag 的值设定为 true，用于弹出文字为“ASP.NET 版 HTML5 测试系统登录成功。”的提示信息。

#### ❑ 用户退出登录页面

在用户身份统一认证系统中单击退出按钮后，会向本系统中的用户退出登录页面发送消



息, 通知本系统将该用户注销。用户退出登录页面中的代码如代码清单 7-22 所示。

代码清单 7-22 用户退出登录页面中的代码

---

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="logoff.aspx.cs"
Inherits="HTML5TEST.logoff" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title>退出</title>
<script language="javascript">
window.addEventListener("message", function (ev) {
    if (ev.origin != "http://localhost:1618") {
        return;
    }
    if (ev.data == "logoff")
        document.getElementById("Logoff_Btn").click();
}, false);
</script>
</head>
<body>
<form id="form1" runat="server">
<asp:Button id="Logoff_Btn" Text="退出" runat="server"
onclick="Logoff_Btn_Click" ClientIDMode="Static"/>
</form>
</body>
</html>
```

---

本页面中只有一个退出按钮。在脚本代码中指定, 如果本系统接收到其他域或其他端口发送过来的数据, 全都不做处理。只有在接收到用户身份统一认证系统中发送过来的数据时才自动单击退出按钮, 执行服务器端的 Logoff\_Btn\_Click 函数, 在服务器端将用户注销。

本页面的服务器端代码如代码清单 7-23 所示。

代码清单 7-23 用户退出登录页面的服务器端代码

---

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace HTML5TEST
{
    public partial class logoff : System.Web.UI.Page
    {
        protected void Logoff_Btn_Click(object sender, EventArgs e)
        {
            Session.Clear();
        }
    }
}
```

---

```

        Response.Redirect("index.aspx?crossDomainFlag=logoff");
    }
}
}

```

在服务器端的 Logoff\_Btn\_Click 函数中将 session 清空，将页面重定向到用户登录页面，指定 URL 参数 crossDomainFlag 的值为 “logoff”，用于在打开用户登录页面后弹出文字为 “ASP.NET 版 HTML5 测试系统成功退出” 的提示信息。

### 3. 用户身份统一认证系统

在用户身份统一认证系统中制作了一个用户登录页面，用户在该页面中输入用户名与密码并单击登录按钮后，可同时自动登录到 Java 版本的 Web 测试系统与 ASP.NET 版本的 Web 测试系统，单击退出按钮可同时从这两个系统中退出。

该页面中使用的控件信息如表 7-3 所示。

表 7-3 用户登录页面中使用的控件信息

控件名称	使用元素	显示文字	说明
log 图片	img		
用户名文字	页面文字	用户名	
用户名文本框	input type="textBox"		最大输入位数为 16
密码文字		密码	
密码文本框	input type="password"		最大输入位数为 16
登录按钮	asp:Button	登录	单击登录按钮后自动登录到 Java 版本的 Web 测试系统与 ASP.NET 版本的 Web 测试系统中
取消按钮	input type="button"	取消	单击取消按钮后关闭页面
退出按钮	input type="button"	退出	单击退出按钮后自动从 Java 版本的 Web 测试系统与 ASP.NET 版本的 Web 测试系统中退出
装载 Java 版本的 Web 测试系统的首页的 iframe	iframe		iframe 中装载 Java 版本的 Web 测试系统的首页页面（即用户登录页面），状态为隐藏状态
装载 ASP.NET 版本的 Web 测试系统的首页的 iframe	iframe		iframe 中装载 ASP.NET 版本的 Web 测试系统的首页页面（即用户登录页面），状态为隐藏状态
装载 Java 版本的 Web 测试系统的退出登录页面的 iframe	iframe		iframe 中装载 Java 版本的 Web 测试系统的退出登录页面，状态为隐藏状态
装载 ASP.NET 版本的 Web 测试系统的退出登录页面的 iframe	iframe		iframe 中装载 ASP.NET 版本的 Web 测试系统的退出登录页面，状态为隐藏状态

用户登录页面的 HTML 页面代码与样式代码如代码清单 7-24 所示。

代码清单 7-24 用户登录页面的 HTML 页面代码与样式代码

```

<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title> 跨域通信示例 </title>
<style type="text/css">
input{
    width: 90px;
    height: 20px;
    font-size:12px;
}
input[type="button"]{
    font-size: 12px;
    width: 68px;
    height: 20px;
    cursor: hand;
    border:none;
    font-family: 宋体 ;
    background-color:White;
    background-image: url(images/but_bg.gif);
    color: white;
}
iframe{
    display:none;
}
body{
    background-image:url(images/backbar.JPG);
}
img{
    width:157px;
    height:52px;
}
</style>
<script type="text/javascript"
... 脚本代码稍后介绍 ...
</script>
</head>
<body>
<form id="form1">
<center>
<table border="0">
    <tr>
        <td colspan="2">
            <div align="center"></div>
        </td>
    </tr>
    <tr>
        <td><font size="3"><b>用户名 </b></font></td>

```

```

        <td>
            <input type="text" id="userName" maxlength="16"
                name="userName" />
        </td>
    </tr>
    <tr>
        <td><font size="3"><b> 密码 </b></font></td>
        <td>
            <input type="password" id="userPass" maxlength="16"
                name="userPass" />
        </td>
    </tr>
    <tr>
        <td colspan="2" align="center">
            <input type="button" value=" 登录 " onclick="Login_Btn_Click()" />
            <input type="button" value=" 取消 " onclick="Cancel_Btn_Click()" />
            <input type="button" value=" 退出 " onclick="Logoff_Btn_Click()" />
        </td>
    </tr>
</table>
</center>
</form>
<iframe src=http://localhost:8443/HTML5 ></iframe>
<iframe src=http://localhost:1116></iframe>
<iframe src=http://localhost:8443/HTML5/logoff.jsp></iframe>
<iframe src=http://localhost:1116/logoff.aspx ></iframe>
</body>
</html>

```

用户登录页面的 JavaScript 脚本代码如代码清单 7-25 所示。

代码清单 7-25 用户登录页面的 JavaScript 脚本代码

```

<script type="text/javascript">
moveTo((screen.width - 400) / 2, (screen.height - 300) / 2);
resizeTo(400, 340);
function Login_Btn_Click() {
    if (document.getElementById("userName").value.trim() == "")
        alert(" 必须输入用户名。");
    else if (document.getElementById("userPass").value.trim() == "")
        alert(" 必须输入密码。");
    else {
        var data = new Object();
        data.userName = document.getElementById("userName").value.trim();
        data.userPass = document.getElementById("userPass").value.trim();
        var iframe = window.frames[0];
        iframe.postMessage(JSON.stringify(data),
            "http://localhost:8443/HTML5");
        var iframe = window.frames[1];
        iframe.postMessage(JSON.stringify(data), "http://localhost:1116");
    }
}

```

```

    }
}
function Cancel_Btn_Click() {
    window.close();
}
function Logoff_Btn_Click() {
    var iframe = window.frames[2];
    iframe.postMessage("logoff",
        "http://localhost:8443/HTML5/logoff.jsp");
    var iframe = window.frames[3];
    iframe.postMessage("logoff", "http://localhost:1116/logoff.aspx");
}
</script>

```

在此段脚本代码的开头，将页面宽度设定为 400 个像素，高度设定为 340 个像素，并将页面移动到屏幕中央。

单击登录按钮时调用 Login\_Btn\_Click 函数，在该函数中先进行检查。如果用户没有输入用户名，则弹出错误提示信息，文字为“必须输入用户名。”；如果用户没有输入密码，则弹出错误提示信息，文字为“必须输入密码。”；如果用户输入了用户名和密码，则将用户名和密码封装在对象中，然后使用 JSON 对象的 stringify 方法将其转换为字符串并发送到 Java 版本的 Web 测试系统的首页与 ASP.NET 版本的 Web 测试系统的首页中，这两个系统在接收到用户名与密码时会自动使用户登录到系统中。

单击取消按钮时会调用 Cancel\_Btn\_Click 函数将页面关闭。

单击退出按钮时将字符串“logoff”发送到 Web 系统的用户退出登录页面与 ASP.NET 版本的 Web 测试系统的用户退出登录页面中，这两个系统在接收到本系统发送出去的“logoff”字符串后会自动将用户从本系统中注销。

## 7.2 案例 24：通过跨文档消息传输功能获取批量数据

### 7.2.1 案例概述

本节介绍如何通过跨文档消息传输功能来获得网络中另一个域、另一个网站、另一个 Web 系统中的一批数据。在某些场合下，这将是一个非常有用的功能（譬如需要对某一个用户在几个系统中的数据进行汇总或比较时）。

### 7.2.2 页面显示效果

首先来看一下本案例的页面在浏览器中的显示效果，页面打开时的显示效果如图 7-12 所示。

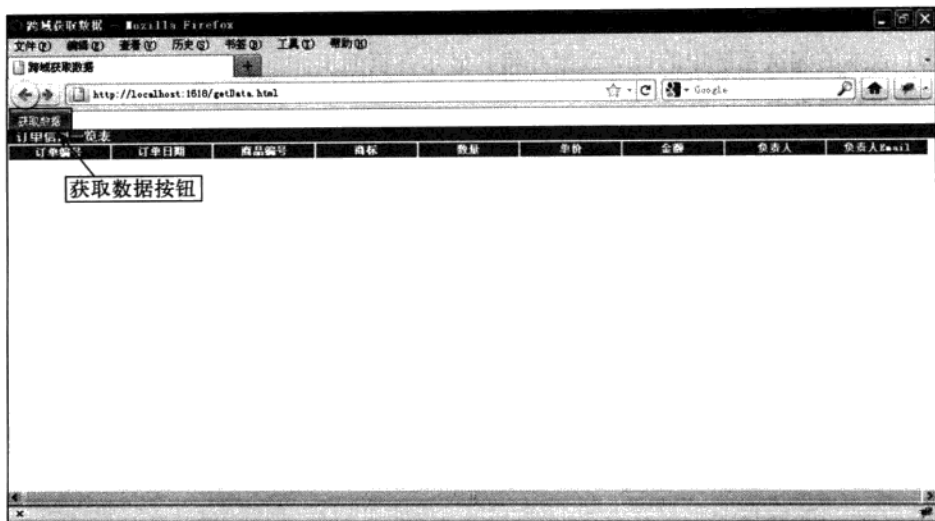


图 7-12 页面打开时的显示效果

单击页面上的获取数据按钮，系统将获取另一个 Web 系统中的一批数据并将其显示在页面上，如图 7-13 所示。

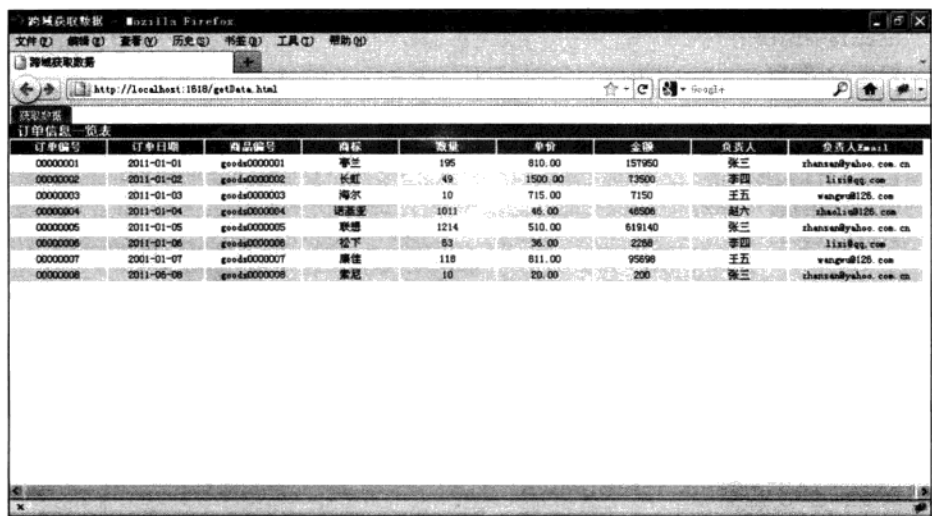


图 7-13 单击获取数据按钮后系统获取另一个 Web 系统中的数据并显示在页面上

### 7.2.3 代码剖析

为了实现本案例的获取另一个 Web 系统中的批量数据的功能，在案例 23 的用户身份统一认证系统（端口为 1618）中添加一个页面（暂且称其为获取数据页面），在该页面中放置

一个获取数据按钮与一个订单信息显示一览表，单击获取数据按钮后系统获取前面介绍的 ASP.NET 版本的测试系统中的订单数据并将其显示在页面上。

### 1. 获取数据页面

获取数据页面的完整代码如代码清单 7-26 所示。

代码清单 7-26 获取数据页面的完整代码

---

```
<!DOCTYPE html>
<html>
<head>
<title> 跨域获取数据 </title>
<style type="text/css">
body {
    margin-left: 0px;
    margin-top: 0px;
}
h1{
    font-size: 14px;
    font-weight: bold;
    color:white;
    background-color:#7088AD;
    text-align:left;
    padding-left:10px;
    display:block;
    width:100%;
    margin:0px;
}
div#buttonDiv{
    text-align:left;
    padding-left:2px;
    width:100%;
}
input [type="button"]{
    font-size: 12px;
    width: 68px;
    height: 20px;
    cursor: hand;
    border:none;
    font-family: 宋体;
    background-color:White;
    background-image: url(images/but_bg.gif);
    color: white;
}
div#infoTable{
    overflow:auto;
    width:100%;
    height:100%;
}
```

```

div#infoTable table{
    width:100%;
    background-color:white;
    cellpadding:1;
    cellspacing:1;
    font-size: 12px;
    text-align: center;
}
div#infoTable table th{
    height:22;
    background-color:#7088AD;
    color: #FFFFFF;
    width:8%;
}
div#infoTable table tr{
    height:30;
}
div#infoTable table tr:nth-child(odd){
    background-color:#E6E6E6;
    color: #333333;
}
div#infoTable table tr:nth-child(even){
    background-color:#fafafa;
    color: black;
}
div#infoTable table tr:nth-child(1){
    background-color:#7088AD;
    color: #FFFFFF;
}
iframe{
    display:none;
}
</style>
<script>
window.addEventListener("message", function (ev) {
    var data;
    if (ev.origin != "http://localhost:1116") {
        return;
    }
    if (ev.data == "数据库连接失败" || ev.data == "读取数据失败")
        alert(ev.data);
    else {
        removeAllData();
        var dataArray = JSON.parse(ev.data);
        for (var i = 0; i < dataArray.length; i++) {
            data = JSON.parse(unescape(dataArray[i]));
            showData(data, i);
        }
    }
}, false);

```



```

function btnGetData_click() {
    var iframe = window.frames[0];
    iframe.postMessage("", "http://localhost:1116/getData.html");
}
function removeAllData() {
    var datatable = document.getElementById("datatable");
    for (var i = datatable.childNodes.length - 1; i > 1; i--) {
        datatable.removeChild(datatable.childNodes[i]);
    }
}
function showData(data) {
    var datatable = document.getElementById("datatable");
    var tr = document.createElement('tr');
    var td1 = document.createElement('td');
    td1.innerHTML = data.Code;
    var td2 = document.createElement('td');
    td2.innerHTML = data.Date;
    var td3 = document.createElement('td');
    td3.innerHTML = data.GoodsCode;
    var td4 = document.createElement('td');
    td4.innerHTML = data.BrandName;
    var td5 = document.createElement('td');
    td5.innerHTML = data.Num;
    var td6 = document.createElement('td');
    td6.innerHTML = data.Price;
    var td7 = document.createElement('td');
    td7.innerHTML = parseInt(data.Num) * parseFloat(data.Price);
    var td8 = document.createElement('td');
    td8.innerHTML = data.PersonName;
    var td9 = document.createElement('td');
    td9.innerHTML = data.Email;
    tr.appendChild(td1);
    tr.appendChild(td2);
    tr.appendChild(td3);
    tr.appendChild(td4);
    tr.appendChild(td5);
    tr.appendChild(td6);
    tr.appendChild(td7);
    tr.appendChild(td8);
    tr.appendChild(td9);
    datatable.appendChild(tr);
}
</script>
</head>
<body>
<section>
    <div id="buttonDiv">
        <input type="button" name="btnGetData" id="btnGetData"
            value=" 获取数据 " onclick="btnGetData_click();" />
    </div>

```

```

</section>
<section>
  <header id="div_head_title_big">
    <h1> 订单信息一览表 </h1>
  </header>
  <div id="infoTable">
    <table id="datatable">
      <tr>
        <th> 订单编号 </th>
        <th> 订单日期 </th>
        <th> 商品编号 </th>
        <th> 商标 </th>
        <th> 数量 </th>
        <th> 单价 </th>
        <th> 金额 </th>
        <th> 负责人 </th>
        <th> 负责人Email</th>
      </tr>
    </table>
  </div>
</section>
<iframe src=http://localhost:1116/getData.html ></iframe>
</body>
</html>

```

在这个页面中，除了放置了一个获取数据按钮及订单信息一览表之外，还放置了一个隐藏的 iframe，以装载 ASP.NET 版本的测试系统的用来发送批量数据的页面（暂且称其为发送数据页面），当用户单击页面上的获取数据按钮后，在 JavaScript 脚本代码中向发送数据页面发送消息，发送数据页面接收到消息后向本页面返回数据库中的全部订单数据，本页面在接收到这批数据后将其显示在页面的订单信息一览表中。

接下来详细分析一下获取数据页面中的 JavaScript 脚本代码。

在脚本代码的开头，定义获取数据页面在接收到其他域或端口传送过来的消息后所要执行的函数。在该函数中，首先规定，如果接收的数据不是 ASP.NET 版本的测试系统（URL 地址为“http://localhost:1116”）发送的数据，则不做任何处理，从而保证了本系统的安全性。如果接收的数据是 ASP.NET 版本的测试系统发送的，而且数据为“数据库连接失败”文字或“读取数据失败”文字（表示 ASP.NET 版本的测试系统连接数据库失败或读取订单信息失败），则直接弹出该错误信息；如果数据不为“数据库连接失败”文字或“读取数据失败”文字，则先调用 removeAllData 函数删除订单信息一览表中的数据（因为用户可能之前已经单击获取数据按钮获取过一批数据），然后调用 JSON 对象的 parse 方法将数据解析为一个数组，之后遍历该数组，将数组中的每一项解析为一个对象，并且调用 showData 函数将该对象中的每一个属性显示在订单信息一览表的每一列中。

在接下来的 JavaScript 脚本代码中，定义了几个本页面中使用的函数。

### ❑ btnGetData\_click 函数

当用户单击页面上的获取数据按钮时调用 btnGetData\_click 函数向页面上隐藏 iframe 中装载的 ASP.NET 版本的测试系统中用来发送批量数据的页面发送消息，通知该页面到数据库中获取全部订单数据。

### ❑ removeAllData 函数

当页面接收到 ASP.NET 版本的测试系统发送的一批订单数据后，先调用 removeAllData 函数将订单信息一览表中当前显示的全部数据进行删除（因为用户之前可能已经单击获取数据按钮获取过一批数据）。

### ❑ showData 函数

当页面接收到 ASP.NET 版本的测试系统中发送过来的数据后，会将该数据解析为一个数组，然后遍历该数组，将数组中的每一项解析为一个对象，然后调用 showData 函数将这个对象中的每一个属性显示在订单信息一览表的每一列中。

showData 函数接收一个 data 参数，该参数代表通过 JSON 对象的 parse 方法将数组中的每一项转换成的对象。在该函数中，为每一个对象在订单信息一览表中添加一行，然后为该对象中的每一个属性添加一列并将该对象的每一个属性显示在订单信息一览表的每一列中。

## 2. 发送数据页面

接下来看一下 ASP.NET 版本的测试系统中用来向用户身份统一认证系统中的获取数据页面发送订单数据的发送数据页面的完整代码，如代码清单 7-27 所示。

代码清单 7-27 发送数据页面的完整代码

---

```
<!DOCTYPE html>
<html>
<head>
<title></title>
<script language="javascript">
window.addEventListener("message", function (ev) {
    if (ev.origin != "http://localhost:1618") {
        return;
    }
    var xhr = new XMLHttpRequest();
    xhr.open("POST", "getDataBackCode.aspx");
    xhr.onreadystatechange =
    function () {
        var result = xhr.responseText;
        if (xhr.readyState == 4) {
            ev.source.postMessage(result, ev.origin);
        }
    }
    xhr.send(null);
}, false);
</script>
</head>
```

```
<body>
</body>
</html>
```

该页面为一个空页面，主要功能是在 JavaScript 脚本代码中定义页面接收到其他域或端口发送的消息后所要执行的处理。在脚本代码中定义，如果系统接收到的消息不是用户身份统一认证系统（URL 地址为“http://localhost:1618”）发送过来的消息，则不执行任何处理。如果系统接收到的消息是用户身份统一认证系统发送过来的消息，则使用 AJAX 后台调用服务器端代码（代码文件为 getDataBackCode.aspx）到数据库中查询全部订单数据，并将查询结果发送回用户身份统一认证系统。

getDataBackCode.aspx 为一个空页面，其主要功能是提供用来查询数据库中全部订单数据的服务器端代码文件 getDataBackCode.aspx.cs，该文件中的内容如代码清单 7-28 所示。

代码清单 7-28 getDataBackCode.aspx.cs 文件中的全部代码

```
< using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Configuration;
using System.Data;
using System.Data.SqlClient;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Json;
using System.IO;
using System.Text;
namespace HTML5TEST
{
    public partial class getDataBackCode : System.Web.UI.Page
    {
        [DataContract]
        public class Data
        {
            [DataMember]
            public String Code { get; set; }
            [DataMember]
            public String Date { get; set; }
            [DataMember]
            public String GoodsCode { get; set; }
            [DataMember]
            public String BrandName { get; set; }
            [DataMember]
            public String Num { get; set; }
            [DataMember]
            public String Price { get; set; }
            [DataMember]
```

```

        public String PersonName { get; set; }
        [DataMember]
        public String Email { get; set; }
    }
    private System.Data.SqlClient.SqlConnection SqlCon;
    private System.Data.SqlClient.SqlDataAdapter DataAdapter;
    string Constr;
    private bool SuccessFlag;
    private DataContractJsonSerializer serializerArray;
    private DataContractJsonSerializer serializer;
    private Data myData;
    protected void Page_Load(object sender, EventArgs e)
    {
        DataContractJsonSerializer serializerArray;
        DataContractJsonSerializer serializer;
        Data myData;
        String strResult = String.Empty;
        String result;
        string SqlStr;
        String orderDate;
        DataSet ds;
        Constr = System.Configuration.ConfigurationManager.
            ConnectionStrings["dbConnection"].ToString();
        this.SqlCon = new System.Data.SqlClient.SqlConnection();
        this.SqlCon.ConnectionString = Constr;
        SuccessFlag = OpenDBConnection();
        if (!SuccessFlag)
        {
            strResult = " 数据库连接失败 ";
            this.WriteReturnStr(strResult);
            return;
        }
        serializerArray =
            new DataContractJsonSerializer(typeof(ArrayList));
        serializer = new DataContractJsonSerializer(typeof(Data));
        SqlStr = "select * from orders order by code";
        ds = this.GetDataFromDB(SqlStr);
        if (ds == null)
        {
            strResult = " 读取数据失败 ";
            this.WriteReturnStr(strResult);
            return;
        }
        using (MemoryStream stream = new MemoryStream())
        {
            ArrayList dataArray = new ArrayList();
            for (int i = 0; i < ds.Tables[0].Rows.Count; i++)
            {
                using (MemoryStream stream1 = new MemoryStream())
                {
                    myData = new Data();
                    myData.Code = ds.Tables[0].Rows[i]["code"].ToString();
                }
            }
        }
    }

```

```

        orderDate=
        ds.Tables[0].Rows[i]["orderDate"].ToString();
        myData.Date =
        DateTime.Parse(orderDate).ToString("yyyy-MM-dd");
        myData.GoodsCode =
        ds.Tables[0].Rows[i]["goodsCode"].ToString();
        myData.BrandName =
        ds.Tables[0].Rows[i]["brandName"].ToString();
        myData.Num = ds.Tables[0].Rows[i]["num"].ToString();
        myData.Price =
        ds.Tables[0].Rows[i]["price"].ToString();
        myData.PersonName =
        ds.Tables[0].Rows[i]["personName"].ToString();
        myData.Email =
        ds.Tables[0].Rows[i]["email"].ToString();
        serializer.WriteObject(stream1, myData);
        result = Encoding.UTF8.GetString(stream1.ToArray());
        dataArray.Add(result);
    }
}
serializerArray.WriteObject(stream, dataArray);
result = Encoding.UTF8.GetString(stream.ToArray());
this.WriteReturnStr(result);
}
}
private bool OpenDBConnection()
{
    try
    {
        this.SqlCon.Open();
        return true;
    }
    catch (Exception e)
    {
        return false;
    }
}
public System.Data.DataSet GetDataFromDB(string SqlStr)
{
    try
    {
        DataSet ds = new DataSet();
        this.DataAdapter = new SqlDataAdapter(SqlStr, this.SqlCon);
        this.DataAdapter.Fill(ds);
        return ds;
    }
    catch (Exception e)
    {
        return null;
    }
}
private void WriteReturnStr(String ReturnStr)

```

```
        {
            Response.Write(ReturnStr);
            Response.Flush();
        }
    }
}
```

在这段代码中，先连接数据库，如果数据库连接失败，则将服务器端返回字符串设定为“数据库连接失败”；接着读取数据库中的全部订单数据，如果读取过程中发生错误，则将服务器端返回字符串设定为“读取数据失败”，否则使用 `DataContractJsonSerializer` 类将读取到的每一行数据进行序列化（在客户端可以使用 JSON 对象的 `parse` 方法将其解析为一个 JavaScript 对象，该对象的每个属性值为在此处设置的每个订单数据的各个字段值），然后将序列化后的结果转换为字符串存放在一个数组中；最后使用 `DataContractJsonSerializer` 类将这个存有所有订单数据的数组进行序列化（在客户端可以使用 JSON 对象的 `parse` 方法将其解析成一个 JavaScript 对象组成的数组），然后将序列化后的结果转换为字符串并将其返回给客户端。

### 7.3 本章小结

本章通过两个案例，详细阐述了 HTML 5 中跨文档消息传输功能的概念及使用方法。在案例 23 中，向读者阐述了如何利用跨文档消息传输功能来实现单点登录功能；在案例 24 中，向读者阐述了如何在一个 Web 系统中利用跨文档消息传输功能来获取位于另一个域或另一个 IP 地址上的 Web 系统中的一批数据。

在案例 23 中，在实现单点登录功能的时候，要求用户在多个 Web 系统中使用相同的用户名与密码，实际上完全可以让用户在不同的 Web 系统中使用不同的用户名与密码，实现方法如下。

首先，在每一个系统的数据库中创建一张存放用户信息的数据表，该表中的字段如表 7-4 所示（该表仅列举了在本章的用户身份统一认证系统、Java 版本的测试系统以及 ASP.NET 版本的测试系统所使用的不同的用户名与密码，在实际使用时可根据需要添加或修改字段名）。

表 7-4 存放用户信息的数据表的字段

字段名	字段类型	字段中存放的值
userName	varchar2(50)	存放用户在用户身份统一认证系统中的用户名
userPass	varchar2(50)	存放用户在用户身份统一认证系统中的密码
userName_Java	varchar2(50)	存放用户在 Java 版测试系统中的用户名
UserPass_Java	varchar2(50)	存放用户在 Java 版测试系统中的密码
userName_ASP	varchar2(50)	存放用户在 ASP.NET 版测试系统中的用户名
UserPass_ASP	varchar2(50)	存放用户在 ASP.NET 版测试系统中的密码

表 7-4 中用户信息的内容如表 7-5 所示。

表 7-5 存放用户信息的数据表中的内容示例

userName	userPass	userName_Java	UserPass_Java	userName_ASP	UserPass_ASP
admin	pass	admin_Java	pass_Java	admin_ASP	pass_ASP

当用户在用户身份统一认证系统中输入用户名与密码并提交表单后，到数据库中查询该用户名与密码是否存在，如果存在则查出该用户在 Java 版测试系统中的用户名与密码，以及在 ASP.NET 版测试系统中的用户名与密码，然后发送消息给这两个系统，通知其实现用户自动登录。

当用户在 Java 版测试系统中输入用户名与密码并且单击登录按钮提交表单后，到数据库中查询该用户名与密码是否存在，如果存在则查出该用户在 ASP.NET 版测试系统中的用户名与密码，然后发送消息给 ASP.NET 版测试系统，通知其实现用户自动登录。

当用户在 ASP.NET 版测试系统中输入用户名与密码并且单击登录按钮提交表单后，到数据库中查询该用户名与密码是否存在，如果存在则查出该用户在 Java 版测试系统中的用户名与密码，然后发送消息给 Java 版测试系统，通知其实现用户自动登录。

另外，虽然从一个 Web 系统向另一个 Web 系统中发送的是未经加密的用户名与密码，但也可以在用户单击登录按钮后进行表单提交，在服务器端将用户名与密码加密，然后将其发送到另一个 Web 系统系统中，另一个 Web 系统在接收到经过加密的用户名与密码后，将表单提交，然后在服务器端先对用户名与密码进行解密，接下来在数据表中查询用户名与密码是否正确，如果正确则使用户登录到本系统中，同时在 session 中设置经过解密后的用户名与密码。

最后，当一个 Web 系统接收到其他域或端口中的 Web 系统发送的数据后，在判断数据来源的时候，使用的是判断数据来源是否为某一个域 + 某一个端口的方法，也可以直接使用判断数据来源是否为某一个 IP 地址或某一个 IP 地址 + 某一个端口的方法，代码如下所示。

```

window.addEventListener("message", function (ev) {
    if (ev.origin != "http://10.214.45.36:1618" &&
        ev.origin != "http:// 10.214.45.36:8443") {
        return;
    }
})

```

下一章将通过两个案例来介绍通过 HTML 5 中的 Web Workers 来实现网页中的多线程处理。





## 第 8 章

# 利用 Web Workers 实现多线程处理

### 本章内容

- ☐ 案例 25：在后台线程中实现对数据库的增删查改操作
- ☐ 案例 26：在后台线程中实现数据的批量插入
- ☐ 本章小结

本章通过两个案例来详细阐述 HTML 5 中关于多线程处理的一个重要功能——利用 Web Workers 来实现 Web 网页上的多线程处理功能。在案例 25 中，利用 Web Workers 在后台线程中实现对数据库的增删查改操作，并在后台线程中生成页面上某个一览表的完整的内部 HTML 代码，然后在前台脚本中直接将这段 HTML 代码输出到页面上。这样处理的目的是希望读者了解到如何在后台线程中生成页面上某个局部区域中的 HTML 代码，然后在前台脚本中直接将这段 HTML 代码输出到页面上。在案例 26 中，在后台线程中实现对数据库的批量插入操作，目的是希望读者通过本案例了解到如何在后台线程中实现对数据库的批量追加与批量更新操作。

## 8.1 案例 25：在后台线程中实现对数据库的增删查改操作

### 8.1.1 案例概述

本案例所实现的功能与第 2 章案例 4 所实现的功能非常类似，用户在页面的订单信息输入表单中输入订单信息，然后单击页面上的追加、修改或删除按钮将输入信息追加、更新到数据库中，或从数据库中删除。打开页面时页面下部的订单信息一览表中显示数据库中的全部订单信息，用户在数据库中增加、修改或删除订单信息后订单信息一览表中会立即重新显示数据库中的全部订单信息。本案例与第 2 章案例 4 的区别在于用户在案例 4 中单击追加、修改或删除按钮执行表单提交时，页面会被刷新，然后在服务器端代码中执行对数据库的追加、修改或删除操作，而在本案例中，当用户单击追加、修改或删除按钮执行表单提交时，页面不会被刷新，虽然使用 AJAX 可以实现不刷新页面而在后台执行服务器端代码的功能，但是只使用 AJAX，仅能实现页面的无刷新处理，如果在后台执行服务器端代码所需时间较长，页面就会停止在不能操作的状态，用户必须等到后台执行完服务器端代码后才能在页面上执行下一步操作，而使用 HTML 5 中的多线程处理，用户不必等到后台服务器端代码执行完毕即可执行下一步操作，比如在表单中继续输入下一条订单信息。另外，在本案例中，取消了案例 4 中 jQuery 验证器的使用，直接使用浏览器自带的对 HTML 5 中新增元素的验证功能。

### 8.1.2 页面显示效果

首先来看一下本案例页面在浏览器中被打开时的效果，如图 8-1 所示。

用户可以在页面上的表单中输入订单信息，如图 8-2 所示。

用户输入订单信息后单击追加按钮，输入的订单信息即被追加到数据库中，同时在页面下部的订单信息一览表中也会立即显示被追加的这条数据，如图 8-3 所示。

单击信息一览表中的某一行，该行数据会被显示在表单的各控件中，如图 8-4 所示。

用户可以在表单中修改该条订单信息，如图 8-5 所示。

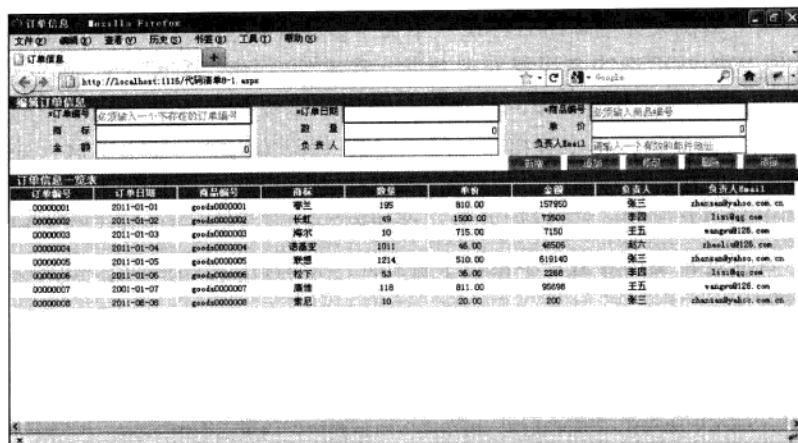


图 8-1 案例页面在浏览器中被打开时的显示效果

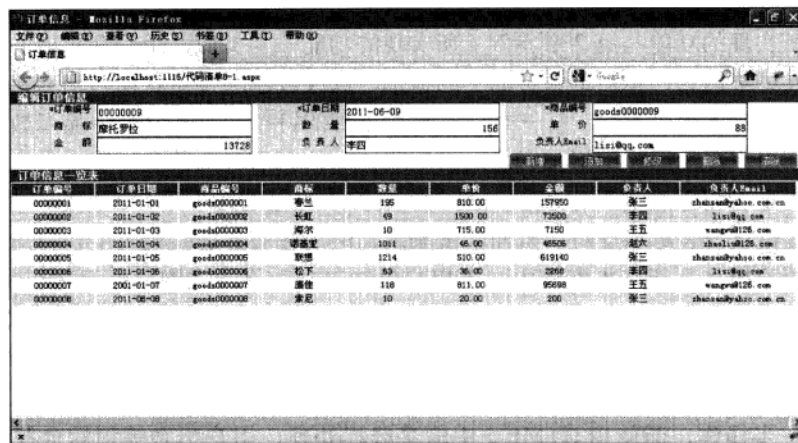


图 8-2 用户在页面上的表单中输入订单信息

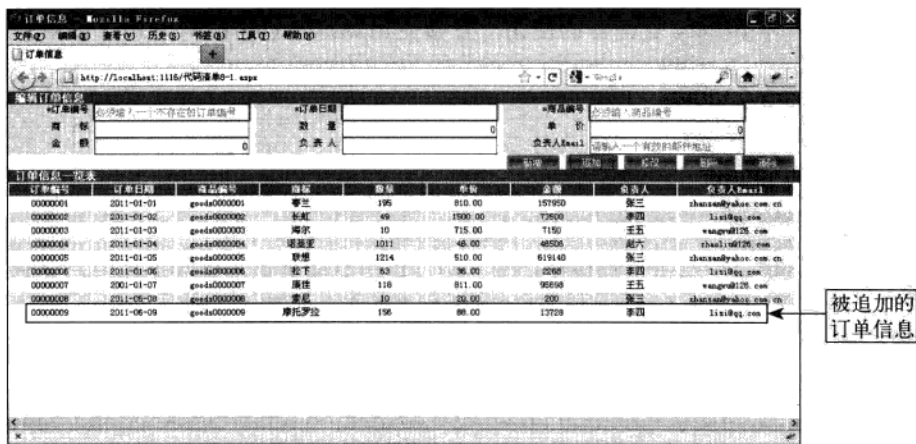


图 8-3 单击追加按钮后表单信息被迫加到数据库中

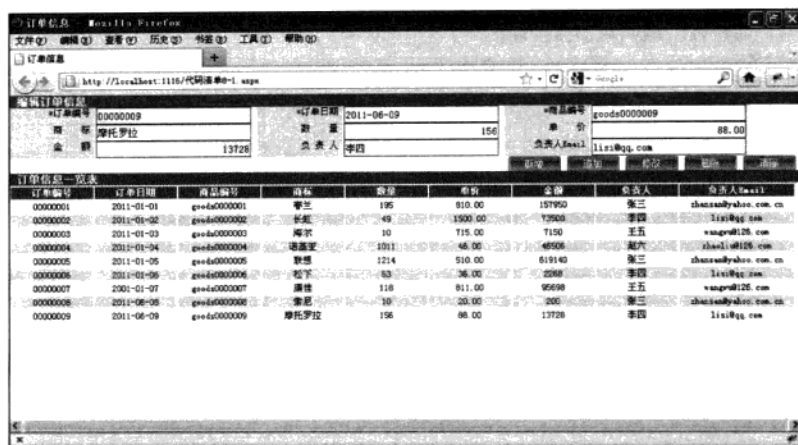


图 8-4 单击信息一览表中的某一行后该行数据会被显示在表单的各控件中

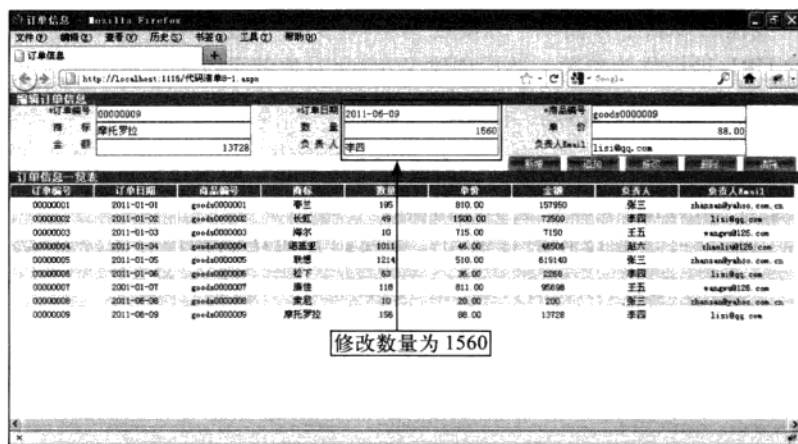


图 8-5 用户可以在表单中修改订单信息

用户修改完订单信息后单击修改按钮，数据库中的该条订单信息将被修改，同时页面上的订单信息一览表中将显示修改后的订单信息，如图 8-6 所示。

另外，如果用户在单击一览表中的某一行使其显示在表单中后再单击删除按钮，则数据库中该条订单信息将被删除，同时一览表中该条数据也将被删除。

### 8.1.3 案例知识点

接下来，在分析案例代码之前先对本案例中使用到的 HTML 5 中的 Web Workers 的有关知识进行介绍。关于 Web Workers 的完整知识，可以参阅笔者所著的《HTML 5 与 CSS 3 权威指南》一书。

首先来看一下，怎样使用 Web Workers API 来创建一个在后台运行的线程。



图 8-6 单击修改按钮后数据库中该条订单信息将被修改

创建后台线程的步骤十分简单。只要在 Worker 类的构造器中将需要在后台线程中执行的脚本文件的 URL 地址作为参数，然后创建 Worker 对象就可以了，示例如下所示。

```
var worker=new Worker("worker.js");
```

要注意，在后台线程中是不能访问页面或窗口对象的。如果在后台线程的脚本文件中使用 window 对象或 document 对象，会引发错误。

另外，可以通过发送和接收消息来与后台线程互相传递数据。通过对 Worker 对象的 onmessage 事件句柄的获取可以在后台线程中接收消息，示例如下所示。

```
worker.onmessage=function(event)
{
    // 处理收到的消息
}
```

使用 Worker 对象的 postMessage 方法对后台线程发送消息，示例如下所示。发送的消息是文本数据，也可以是任何 JavaScript 对象（需要通过 JSON 对象的 stringify 方法将其转换成文本数据）。

```
worker.postMessage(message);
```

另外，同样可以通过获取 Worker 对象的 onmessage 事件句柄及 postMessage 方法在后台线程内部进行消息的接收和发送。

## 8.1.4 代码剖析

### 1. HTML 5 页面代码

该页面中所使用的元素与第 2 章案例 4 中案例页面的元素大致相同，只是取消了 jQuery 验证器的使用，同时将按钮修改为表 8-1 中所描述的按钮。

表 8-1 案例页面中所使用的按钮

按钮名称	元素类型	按钮文字	按钮 ID	执行函数的代码	说明
新增按钮	input type="button"	新增	btnNew	onclick="btnNew_ onclick()"	单击新增按钮后表单中 各控件内元素被清空
追加按钮	input type="submit"	追加	btnAdd	formation="javascript: btnAdd_onclick()"	单击追加按钮后表单信 息被追加到数据库中
修改按钮	input type="submit"	修改	btnUpdate	formation="javascript: btnUpdate_onclick()"	单击修改按钮后用户在 数据库中选择的订单信息 被修改
删除按钮	input type="submit"	删除	btnDelete	formation="javascript: btnDelete_onclick()"	单击删除按钮后用户 在数据库中选择的订单 被删除
清除按钮	input type="button"	清除	btnClear	onclick=" btnClear_ onclick()"	单击清除按钮后表单 中除订单编号输入框之 外的其他文本框中的内 容被清除

经过上述修改之后, 表单的 HTML 代码如代码清单 8-1 所示。案例页面中的其他 HTML 代码与整个页面的样式代码与第 2 章案例 4 中的 HTML 代码与样式代码完全相同, 故不再赘述。

代码清单 8-1 表单的 HTML 代码

```

<form id="form1" method="post" runat=server>
<ul>
  <li>
    <ul>
      <li id="title_1">
        <span>*</span><label for="tbxCde"> 订单编号 </label>
      </li>
      <li id="content_1">
        <input type="text" id="tbxCde" name="tbxCde" maxlength="8"
        placeholder=" 必须输入一个不存在的订单编号 " autofocus required/>
      </li>
      <li id="title_2">
        <span>*</span><label for="tbxDate"> 订单日期 </label>
      </li>
      <li id="content_2">
        <input type="date" id="tbxDate" name="tbxDate" maxlength="10"
        required/>
      </li>
      <li id="title_3">
        <span>*</span><label for="tbxGoodsCode"> 商品编号 </label>
      </li>
      <li id="content_3">
        <input type="text" id="tbxGoodsCode" name="tbxGoodsCode"
        maxlength="12" placeholder=" 必须输入商品编号 " required/>
      </li>
    </ul>
  </li>
</ul>

```

[illegible]

```

    </ul>
    </li>
</ul>
<div id="buttonDiv">
    <input type="button" name="btnNew" id="btnNew" value="新增"
    onclick="btnNew_onclick();" />
    <input type="submit" name="btnAdd" id="btnAdd" value="追加"
    formaction="javascript:btnAdd_onclick();" />
    <input type="submit" name="btnUpdate" id="btnUpdate" value="修改"
    disabled formaction="javascript:btnUpdate_onclick();" />
    <input type="submit" name="btnDelete" id="btnDelete" value="删除"
    disabled formaction="javascript:btnDelete_onclick();" />
    <input type="button" name="btnClear" id="btnClear" value="清除"
    onclick="btnClear_onclick();" />
</div>
</form>

```

---

## 2. 页面上的 JavaScript 脚本代码

接下来看一下本案例页面中的 JavaScript 脚本代码，如代码清单 8-2 所示。

代码清单 8-2 案例页面中的 JavaScript 脚本代码

---

```

<script>
var data;
var datatable;
function window_onload() {
    var worker = new Worker("代码清单 8-2Worker.js");
    worker.postMessage("load");
    worker.onmessage = function (event) {
        if (event.data == "数据库连接失败" || event.data == "读取数据失败")
            alert(event.data);
        else
        {
            datatable = document.getElementById("datatable");
            datatable.innerHTML = event.data;
        }
    }
}
function tbxNum_onblur()
{
    var num, price;
    num = parseInt(document.getElementById("tbxNum").value);
    price = parseFloat(document.getElementById("tbxPrice").value);
    if (isNaN(num * price)) {
        document.getElementById("tbxNum").value = "0";
        document.getElementById("tbxMoney").value = "0";
    }
    else
        document.getElementById("tbxMoney").value = num * price;
}

```



```

function tbxPrice_onblur()
{
    var num,price;
    num=parseInt (document.getElementById("tbxNum").value);
    price=parseFloat (document.getElementById("tbxPrice").value);
    if (isNaN(num*price))
    {
        document.getElementById("tbxPrice").value="0";
        document.getElementById("tbxMoney").value="0";
    }
    else
        document.getElementById("tbxMoney").value= num * price;
}

function btnAdd_onclick() {
    data = new Object();
    data.operateType = "add";
    data.Code = document.getElementById("tbxCode").value;
    data.Date = document.getElementById("tbxDate").value;
    data.GoodsCode = document.getElementById("tbxGoodsCode").value;
    data.BrandName = document.getElementById("tbxBrandName").value;
    data.Num = document.getElementById("tbxNum").value;
    data.Price = document.getElementById("tbxPrice").value;
    data.PersonName = document.getElementById("tbxPersonName").value;
    data.Email = document.getElementById("tbxEmail").value;
    var worker = new Worker(" 代码清单 8-2Worker.js");
    worker.postMessage(JSON.stringify(data));
    worker.onmessage = function (event) {
        datatable = document.getElementById("datatable");
        if (event.data == "数据库连接失败" || event.data == "读取数据失败" ||
            event.data == "数据添加失败" || event.data == "输入的订单编号在数据库
            中已存在")
            alert(event.data);
        else {
            datatable.innerHTML = event.data;
            alert(" 成功添加数据!");
            btnNew_onclick();
        }
    }
}

function btnUpdate_onclick() {
    data = new Object();
    data.operateType = "update";
    data.Code = document.getElementById("tbxCode").value;
    data.Date = document.getElementById("tbxDate").value;
    data.GoodsCode = document.getElementById("tbxGoodsCode").value;
    data.BrandName = document.getElementById("tbxBrandName").value;
    data.Num = document.getElementById("tbxNum").value;
    data.Price = document.getElementById("tbxPrice").value;
    data.PersonName = document.getElementById("tbxPersonName").value;

```

```

data.Email = document.getElementById("tbxEmail").value;
var worker = new Worker("代码清单 8-2Worker.js");
worker.postMessage(JSON.stringify(data));
worker.onmessage = function (event) {
    datatable = document.getElementById("datatable");
    if (event.data == "数据库连接失败" || event.data == "读取数据失败" ||
        event.data == "数据修改失败")
        alert(event.data);
    else {
        datatable.innerHTML = event.data;
        alert("成功修改数据!");
    }
}
}
function btnDelete_onclick()
{
    data = new Object();
    data.operateType = "delete";
    data.Code = document.getElementById("tbxCode").value;
    var worker = new Worker("代码清单 8-2Worker.js");
    worker.postMessage(JSON.stringify(data));
    worker.onmessage = function (event) {
        datatable = document.getElementById("datatable");
        if (event.data == "数据库连接失败" || event.data == "读取数据失败" ||
            event.data == "数据删除失败")
            alert(event.data);
        else {
            datatable.innerHTML = event.data;
            alert("成功删除数据!");
            btnNew_onclick();
        }
    }
}
function btnNew_onclick() {
    document.getElementById("form1").reset();
    document.getElementById("tbxCode").removeAttribute("readonly");
    document.getElementById("btnAdd").disabled="";
    document.getElementById("btnUpdate").disabled="disabled";
    document.getElementById("btnDelete").disabled="disabled";
}
function btnClear_onclick()
{
    document.getElementById("tbxDate").value="";
    document.getElementById("tbxGoodsCode").value="";
    document.getElementById("tbxBrandName").value="";
    document.getElementById("tbxNum").value="0";
    document.getElementById("tbxPrice").value="0";
    document.getElementById("tbxMoney").value="0";
    document.getElementById("tbxPersonName").value="";
    document.getElementById("tbxEmail").value="";
}

```

```

}
function tr_onclick(tr,i)
{
    document.getElementById("tbxCode").value=
    tr.children.item(0).innerHTML;
    document.getElementById("tbxDate").value=
    tr.children.item(1).innerHTML;
    document.getElementById("tbxGoodsCode").value=
    tr.children.item(2).innerHTML;
    document.getElementById("tbxBrandName").value=
    tr.children.item(3).innerHTML;
    document.getElementById("tbxNum").value=
    tr.children.item(4).innerHTML;
    document.getElementById("tbxPrice").value=
    tr.children.item(5).innerHTML;
    document.getElementById("tbxMoney").value=
    tr.children.item(6).innerHTML;
    document.getElementById("tbxPersonName").value=
    tr.children.item(7).innerHTML;
    document.getElementById("tbxEmail").value=
    tr.children.item(8).innerHTML;
    document.getElementById("tbxCode").setAttribute("readonly",true);
    document.getElementById("btnAdd").disabled="disabled";
    document.getElementById("btnUpdate").disabled="";
    document.getElementById("btnDelete").disabled="";
}
</script>

```

在脚本代码的开始处，定义了两个全局变量，其中 data 变量代表一个 JavaScript 对象，该对象的各属性值中保存了一条订单信息的订单编号、订单日期等数据。Datatable 变量代表页面中的订单信息一览表。

下面介绍脚本代码中的各函数定义。

#### □ window\_onload 函数

window\_onload 函数在打开页面时被调用。该函数先指定后台脚本代码文件的文件名为“代码清单 8-2Worker.js”，然后向该后台脚本代码文件发送消息，消息内容为一个“load”字符串，该字符串决定在后台脚本代码中执行从数据库中读取所有订单信息并创建显示所有订单信息的订单信息一览表的 HTML 页面代码。在接收到该后台脚本代码传回的消息后进行判断，如果传回的消息文字为“数据库连接失败”或“读取数据失败”，则直接弹出错误信息提示框，提示信息文字为从后台脚本代码传回来的错误消息文字；否则将订单信息一览表的 HTML 代码设定为后台脚本代码中传回的 HTML 页面代码。

#### □ tbxNum\_onblur 函数

当页面上的数量文本框失去焦点时调用 tbxNum\_onblur 函数。在该函数中进行判断，如果数量文本框中的输入值乘以单价文本框中的输入值后得到的结果不是有效数值，则将数量文本框与金额文本框中的内容均设定为 0；否则将金额文本框中的内容设定为数量文本框中

的输入值乘以单价文本框中的输入值后得到的结果。

#### ❑ tbxPrice\_onblur 函数

当页面上的单价文本框失去焦点时调用 tbxPrice\_onblur 函数。在该函数中进行判断，如果数量文本框中的输入值乘以单价文本框中的输入值后得到的结果不是有效数值，则将单价文本框与金额文本框中的内容均设定为 0，否则将金额文本框中的内容设定为数量文本框中的输入值乘以单价文本框中的输入值后得到的结果。

#### ❑ btnAdd\_onclick 函数

当用户单击追加按钮时调用 btnAdd\_onclick 函数。该函数先利用变量 data 创建一个 JavaScript 对象，该对象的 operateType 属性值为“add”，该属性值决定在后台脚本代码中执行将 data 对象的其他属性值中的内容用于一条订单信息的各部分内容，然后将该订单信息追加到数据库中。接下来该函数将页面表单上各控件中的内容保存在 data 对象的其他属性值中，然后指定后台脚本代码文件的文件名为“代码清单 8-2Worker.js”，使用 JSON 对象的 stringify 方法将 data 对象转换为字符串，之后将该字符串作为消息发送到后台脚本代码文件中。当接收到该后台脚本代码传回的消息后进行判断，如果传回的消息文字为“数据库连接失败”、“读取数据失败”、“数据添加失败”或“输入的订单编号在数据库中已存在”，则直接弹出错误信息提示框，提示信息文字为从后台脚本代码传回的错误消息文字；否则将订单信息一览表的 HTML 代码设定为后台脚本代码中传回的 HTML 页面代码，然后弹出提示信息，文字为“成功添加数据！”。最后调用 btnNew\_onclick 函数将页面上所有输入文本框中的内容清除。

#### ❑ btnUpdate\_onclick 函数

当用户单击修改按钮时调用 btnUpdate\_onclick 函数。该函数首先使用 data 变量创建一个 JavaScript 对象，该对象的 operateType 属性值为“update”，该属性值决定在后台脚本代码中执行将 data 对象的其他属性值中的内容用于一条订单信息的各部分内容，然后将该订单信息更新到数据库中。接下来将页面表单中各控件中的内容保存在 data 对象的其他属性值中。然后指定后台脚本代码文件的文件名为“代码清单 8-2Worker.js”，使用 JSON 对象的 stringify 方法将 data 对象转换为字符串，之后将该字符串作为消息发送到后台脚本代码文件中。在接收到该后台脚本代码传回的消息后进行判断，如果传回的消息文字为“数据库连接失败”、“读取数据失败”或“数据修改失败”，则直接弹出错误信息提示框，提示信息文字为从后台脚本代码传回的错误消息文字；否则将订单信息一览表的 HTML 代码设定为后台脚本代码中传回的 HTML 页面代码，随后弹出提示信息为“成功修改数据！”。

#### ❑ btnDelete\_onclick 函数

当用户单击删除按钮时调用 btnDelete\_onclick 函数。该函数先使用 data 变量创建一个 JavaScript 对象，该对象的 operateType 属性值为“delete”，该属性值决定在后台脚本代码中执行将数据库中订单编号等于 data 对象的 Code 属性值的订单数据删除。接下来将页面上订单编号文本框中的内容保存在 data 对象的 Code 属性值中。然后指定后台脚本代码文件的文件名为“代码清单 8-2Worker.js”，使用 JSON 对象的 stringify 方法将 data 对象转换为字符

串，之后将该字符串作为消息发送到后台脚本代码文件中。在接收到该后台脚本代码传回的消息后进行判断，如果传回来的消息文字为“数据库连接失败”、“读取数据失败”或“数据删除失败”，则直接弹出错误信息提示框，提示信息文字为从后台脚本代码传回的错误消息文字；否则将订单信息一览表的 HTML 代码设定为后台脚本代码中传回的 HTML 页面代码，随后弹出提示信息为“成功删除数据！”。最后调用 btnNew\_onclick 函数将页面上所有输入文本框中的内容清除。

#### □ btnNew\_onclick 函数

当用户单击新增按钮时将调用 btnNew\_onclick 函数。该函数首先调用页面表单的 reset 方法将表单中所有文本框中的内容进行清除，然后将订单编号设定为非只读状态，将追加按钮设定为有效状态，将修改按钮与删除按钮设定为无效状态。

#### □ btnClear\_onclick 函数

当用户单击清除按钮时将调用 btnClear\_onclick 函数。该函数将表单中除了订单编号文本框之外的所有文本框中的内容清除。

#### □ tr\_onclick 函数

当用户单击订单信息一览表中的某一行时将调用 tr\_onclick 函数。该函数接受两个参数，其中 tr 参数为订单信息一览表中被单击的行元素本身，i 参数表示被单击的行是一览表中的第几行（行标题不被统计在内，计数从 0 开始，即单击一览表中数据行的第一行时 i 值为 0）。

该函数首先将被单击行的各列数据显示在表单内的各控件中，然后设定订单编号文本框为只读状态，追加按钮为无效状态，修改按钮与删除按钮均为有效状态。

### 3. 在后台线程中运行的 JavaScript 脚本代码

接下来分析一下本案例中使用的在后台线程中运行的 JavaScript 脚本代码，如代码清单 8-3 所示，该脚本代码的文件名为“代码清单 8-2Worker.js”。

代码清单 8-3 后台线程中运行的 JavaScript 脚本代码

---

```
onmessage = function (event) {
    if (event.data == "load") {
        var xhr = new XMLHttpRequest();
        xhr.open("POST", "代码清单 8-2BackCode.aspx?operateType=load");
        xhr.onreadystatechange =
            function () {
                var result = xhr.responseText;
                if (xhr.readyState == 4) {
                    if (result == "数据库连接失败" || result == "读取数据失败")
                        postMessage(result);
                    else
                        postMessage(createHtml(result));
                }
            }
        xhr.send(null);
    }
}
```

```

else {
    var tmpObj = JSON.parse(event.data);
    if (tmpObj.operateType == "add" || tmpObj.operateType == "update")
    {
        var data = new Object();
        data.Code = tmpObj.Code;
        data.Date = tmpObj.Date;
        data.GoodsCode = tmpObj.GoodsCode;
        data.BrandName = tmpObj.BrandName;
        data.Num = tmpObj.Num;
        data.Price = tmpObj.Price;
        data.PersonName = tmpObj.PersonName;
        data.Email = tmpObj.Email;
        var xhr = new XMLHttpRequest();
        xhr.open("POST", "代码清单 8-2BackCode.aspx?operateType=" +
            tmpObj.operateType);
        xhr.onreadystatechange =
        function () {
            var result = xhr.responseText;
            if (xhr.readyState == 4) {
                if (result == "数据库连接失败" ||
                    result == "读取数据失败" || result == "数据添加失败" ||
                    result == "输入的订单编号在数据库中已存在" ||
                    result == "数据修改失败")
                    postMessage(result);
                else
                    postMessage(createHtml(result));
            }
        }
        xhr.send(JSON.stringify(data));
    }
}
else {
    var tmpObj = JSON.parse(event.data);
    var xhr = new XMLHttpRequest();
    xhr.open("POST", "代码清单 8-2BackCode.aspx?operateType=" +
        tmpObj.operateType + "&code=" + tmpObj.Code);
    xhr.onreadystatechange =
    function () {
        var result = xhr.responseText;
        if (xhr.readyState == 4) {
            if (event.data == "数据库连接失败" ||
                event.data == "读取数据失败" ||
                event.data == "数据删除失败")
                postMessage(result);
            else
                postMessage(createHtml(result));
        }
    }
    xhr.send(null);
}
}

```

```

}
function createHtml(inputData) {
    var html, data;
    html = "<tr>";
    html += "<th> 订单编号 </th>";
    html += "<th> 订单日期 </th>";
    html += "<th> 商品编号 </th>";
    html += "<th> 商标 </th>";
    html += "<th> 数量 </th>";
    html += "<th> 单价 </th>";
    html += "<th> 金额 </th>";
    html += "<th> 负责人 </td>";
    html += "<th> 负责人 Email</th>";
    html += "</tr>";
    var dataArray = JSON.parse(unescape(inputData));
    for (var i = 0; i < dataArray.length; i++) {
        data = JSON.parse(unescape(dataArray[i]));
        html += "<tr onclick='tr_onclick(this,\" + i + \")'>";
        html += "<td>" + data.Code + "</td>";
        html += "<td>" + data.Date + "</td>";
        html += "<td>" + data.GoodsCode + "</td>";
        html += "<td>" + data.BrandName + "</td>";
        html += "<td>" + data.Num + "</td>";
        html += "<td>" + data.Price + "</td>";
        html += "<td>" + parseInt(data.Num) * parseFloat(data.Price) +
            "</td>";
        html += "<td>" + data.PersonName + "</td>";
        html += "<td>" + data.Email + "</td>";
        html += "</tr>";
    }
    return html;
}

```

在后台线程中运行的 JavaScript 脚本代码（即代码清单 8-2Worker.js 文件中的 JavaScript 脚本代码）中定义，当脚本代码接收到案例页面的 JavaScript 脚本代码中发送的消息字符串后进行判断，如果该消息为“load”字符串，则通过 AJAX 调用服务器端 ASP.NET 脚本代码，文件名为“代码清单 8-2BackCode.aspx”。调用服务器端代码时传入一个参数 operateType，参数值为 load。服务器端脚本代码被调用后到数据库中查询所有订单信息并将其作为 JavaScript 数组返回，每一个数组项中保存了数据库中的一条订单信息。脚本代码在接收到服务器端的返回字符串后进行判断，如果返回字符串文字为“数据库连接失败”或“读取数据失败”，则将该文字作为消息字符串传回给案例页面中的 JavaScript 脚本代码，该脚本代码将直接在页面上弹出提示信息，内容为后台线程中运行的 JavaScript 脚本代码的传回字符串（即“数据库连接失败”或“读取数据失败”）。如果服务器端传回一个由数组转换而来的消息字符串，则调用 createHtml 函数创建页面中订单信息一览表的 HTML 代码，然后将这段 HTML 代码作为消息字符串传回给案例页面中的 JavaScript 脚本代码。

在 `createHtml` 函数中使用一个传入参数 `inputData`，该参数为服务器端返回的字符串。在函数内部首先创建案例页面中所有列标题的 HTML 代码，然后通过 JSON 对象的 `parse` 方法将服务器端返回的字符串解析为一个 JavaScript 数组，之后对该数组进行遍历，通过 JSON 对象的 `parse` 方法将数组中的每一项解析为一个 JavaScript 对象，根据数组中所有 JavaScript 对象创建页面订单信息一览表中所有数据行的 HTML 代码，每一行的各列中显示订单信息的订单编号、订单日期等，最后将生成的 HTML 代码返回。

如果在后台线程中运行的 JavaScript 脚本代码接收到的前台页面的 JavaScript 脚本代码发送的消息不是“load”字符串，则将该消息字符串通过 JSON 对象的 `parse` 方法转换为 JavaScript 对象，并赋值给 `tmpObj` 变量。如果 `tmpObj` 对象中 `operateType` 的属性值为“add”或“update”字符串，则创建一个 JavaScript 对象并将其赋值给 `data` 变量。将 `tmpObj` 对象中的其他各属性值（这些属性值中保存了表单上各控件中的内容）赋值给 `data` 对象的各属性值。然后通过 AJAX 调用服务器端 ASP.NET 脚本代码，文件名为“代码清单 8-2BackCode.aspx”。调用服务器端代码时传入一个参数 `operateType`，参数值为 `tmpObj` 对象中 `operateType` 的属性值（“add”或“update”字符串）。同时将 `data` 对象提交给该服务器端 ASP.NET 脚本代码。服务器端 ASP.NET 脚本代码被调用后根据传入的 `operateType` 参数值将 `data` 对象中保存的一条订单信息追加或更新到数据库中。追加或更新成功后重新查询数据库中所有订单信息并将其作为 JavaScript 数组返回。如果追加、更新或查询失败，则将相应的错误信息文字返回。后台脚本代码在接收到服务器端的返回字符串后进行判断，如果返回字符串文字为“数据库连接失败”、“读取数据失败”、“数据添加失败”、“输入的订单编号在数据库中已存在”或“数据修改失败”，则将该文字作为消息字符串传回给案例页面中的 JavaScript 脚本代码，该脚本代码将直接在页面上弹出提示信息，信息的内容为后台线程中运行的 JavaScript 脚本代码传回的字符串（即“数据库连接失败”、“读取数据失败”、“数据添加失败”、“输入的订单编号在数据库中已存在”或“数据修改失败”文字）。如果服务器端传回一个由数组转换而来的消息字符串，则调用 `createHtml` 函数创建页面上订单信息一览表的 HTML 代码，然后将这段 HTML 代码作为消息字符串传回给案例页面中的 JavaScript 脚本代码。

如果 `tmpObj` 对象中的 `operateType` 属性值为“delete”字符串，则通过 AJAX 调用服务器端 ASP.NET 脚本代码，文件名为“代码清单 8-2BackCode.aspx”。调用服务器端代码时传入两个参数：`operateType` 与 `code`，`operateType` 参数的参数值为 `tmpObj` 对象中的 `operateType` 属性值（“add”或“update”字符串），`code` 参数的参数值为 `tmpObj` 对象中的 `Code` 属性值（页面表单中订单编号文本框中的内容）。服务器端 ASP.NET 脚本代码被调用后将数据库中订单编号等于传入的 `code` 值的订单信息删除。删除成功后重新查询数据库中所有订单信息并将其作为 JavaScript 数组返回。如果删除失败或查询数据失败，则将相应的错误信息文字返回。后台脚本代码在接收到服务器端的返回字符串后进行判断，如果返回字符串文字为“数据库连接失败”、“读取数据失败”或“数据删除失败”，则将该文字作为消息字符串传回给案例页面中的 JavaScript 脚本代码，该脚本代码将直接在页面上弹出提示信息，



内容为后台线程中运行的 JavaScript 脚本代码传回的字符串（即“数据库连接失败”、“读取数据失败”或“数据删除失败”文字）。如果服务器端传回一个由数组转换而来的消息字符串，则调用 createHtml 函数创建页面上订单信息一览表的 HTML 代码，然后将这段 HTML 代码作为消息字符串传回给案例页面中的 JavaScript 脚本代码。

#### 4. ASP.NET 服务器端代码

接下来看一下本案例中使用的 ASP.NET 服务器端脚本代码，如代码清单 8-4 所示，该服务器端脚本代码的文件名为“代码清单 8-2BackCode.aspx”。

代码清单 8-4 本案例使用的 ASP.NET 服务器端脚本代码

---

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Configuration;
using System.Data;
using System.Data.SqlClient;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Json;
using System.IO;
using System.Text;
namespace HTML5TEST
{
    public partial class 代码清单 8_1BackCode : System.Web.UI.Page
    {
        [DataContract]
        public class Data
        {
            [DataMember]
            public String Code { get; set; }
            [DataMember]
            public String Date { get; set; }
            [DataMember]
            public String GoodsCode { get; set; }
            [DataMember]
            public String BrandName { get; set; }
            [DataMember]
            public String Num { get; set; }
            [DataMember]
            public String Price { get; set; }
            [DataMember]
            public String PersonName { get; set; }
            [DataMember]
            public String Email { get; set; }
        }
    }
}
```

```

}
private System.Data.SqlClient.SqlConnection SqlCon;
private System.Data.SqlClient.SqlDataAdapter DataAdapter;
private System.Data.SqlClient.SqlCommand Command;//SQL Command
string Constr;
private bool SuccessFlag;
private DataContractJsonSerializer serializerArray;
private DataContractJsonSerializer serializer;
private Data myData;
protected void Page_Load(object sender, EventArgs e)
{
    String strResult = String.Empty;
    String result,operateType;
    string SqlStr;
    DataSet ds;
    Constr = System.Configuration.ConfigurationManager.
        ConnectionStrings["dbConnection"].ToString();
    this.SqlCon = new System.Data.SqlClient.SqlConnection();
    this.SqlCon.ConnectionString = Constr;
    SuccessFlag = OpenDBConnection();
    if (!SuccessFlag)
    {
        strResult = "数据库连接失败";
        this.WriteReturnStr(strResult);
        return;
    }
    operateType= Request.QueryString["operateType"].ToString();
    if (operateType.Equals("load"))
    {
        LoadAllData();
    }
    else if (operateType.Equals("add"))
    {
        ReadData();
        SqlStr = "select count(*) count from orders where code='" +
            myData.Code.Trim().Replace("'", "''") + "'";
        ds = this.GetDataFromDB(SqlStr);
        if (ds == null)
        {
            strResult = "数据添加失败";
            this.WriteReturnStr(strResult);
            return;
        }
        if (Int32.Parse(ds.Tables[0].Rows[0]["count"].ToString())>0)
        {
            strResult = "输入的订单编号在数据库中已存在";
            this.WriteReturnStr(strResult);
            return;
        }
        SqlStr = "insert into orders ";
    }
}

```

```

        SqlStr += "values('" +
        myData.Code.Trim().Replace("'", "''") + "','" +
        SqlStr += "'" + myData.Date.Trim() + "','" +
        SqlStr += "'" +
        myData.GoodsCode.Trim().Replace("'", "''") + "','" +
        SqlStr += "'" +
        myData.BrandName.Trim().Replace("'", "''") + "','" +
        SqlStr += myData.Num.ToString() + "','" +
        SqlStr += myData.Price.ToString() + "','" +
        SqlStr += "'" +
        myData.PersonName.Trim().Replace("'", "''") + "','" +
        SqlStr += "'" + myData.Email.Trim().Replace("'", "''") + "'"");
        SuccessFlag = this.ExecSingleSql(SqlStr);
        if (SuccessFlag == false)
        {
            strResult = "数据添加失败";
            this.WriteReturnStr(strResult);
            return;
        }
        LoadAllData();
    }
    else if (operateType.Equals("update"))
    {
        ReadData();
        SqlStr = "update orders ";
        SqlStr += "set orderDate='" +
        myData.Date.Trim().Replace("'", "''") + "','" +
        SqlStr += "goodsCode='" +
        myData.GoodsCode.Trim().Replace("'", "''") + "','" +
        SqlStr += "brandName='" +
        myData.BrandName.Trim().Replace("'", "''") + "','" +
        SqlStr += "num=" + myData.Num + "','" +
        SqlStr += "price=" + myData.Price + "','" +
        SqlStr += "personName='" +
        myData.PersonName.Trim().Replace("'", "''") + "','" +
        SqlStr += "email='" +
        myData.Email.Trim().Replace("'", "''") + "'"";
        SqlStr += "where code='" +
        myData.Code.Trim().Replace("'", "''") + "'";
        SuccessFlag = this.ExecSingleSql(SqlStr);
        if (SuccessFlag == false)
        {
            strResult = "数据修改失败";
            this.WriteReturnStr(strResult);
            return;
        }
        LoadAllData();
    }
    else if (operateType.Equals("delete"))
    {

```

```

        SqlStr = "delete from orders ";
        SqlStr += "where code='" +
        Request.QueryString["code"].ToString().Trim().
        Replace("'", "''") + "'";
        SuccessFlag = this.ExecSingleSql(SqlStr);
        if (SuccessFlag == false)
        {
            strResult = "数据删除失败";
            this.WriteReturnStr(strResult);
            return;
        }
        LoadAllData();
    }
}

private void LoadAllData()
{
    String strResult = String.Empty;
    String result;
    DataSet ds;
    string SqlStr;
    string OrderDate;
    serializerArray = new DataContractJsonSerializer(
        typeof(ArrayList));
    serializer = new DataContractJsonSerializer(typeof(Data));

    SqlStr = "select * from orders order by code";
    ds = this.GetDataFromDB(SqlStr);
    if (ds == null)
    {
        strResult = "读取数据失败";
        this.WriteReturnStr(strResult);
        return;
    }
    using (MemoryStream stream = new MemoryStream())
    {
        ArrayList dataArray = new ArrayList();
        for (int i = 0; i < ds.Tables[0].Rows.Count; i++)
        {
            using (MemoryStream stream1 = new MemoryStream())
            {
                myData = new Data();
                myData.Code = ds.Tables[0].Rows[i]["code"].ToString();
                OrderDate =
                ds.Tables[0].Rows[i]["orderDate"].ToString();
                myData.Date =
                DateTime.Parse(OrderDate).ToString("yyyy-MM-dd");
                myData.GoodsCode =
                ds.Tables[0].Rows[i]["goodsCode"].ToString();
                myData.BrandName =
                ds.Tables[0].Rows[i]["brandName"].ToString();
            }
        }
    }
}

```

```

        myData.Num = ds.Tables[0].Rows[i]["num"].ToString();
        myData.Price =
        ds.Tables[0].Rows[i]["price"].ToString();
        myData.PersonName =
        ds.Tables[0].Rows[i]["personName"].ToString();
        myData.Email =
        ds.Tables[0].Rows[i]["email"].ToString();
        serializer.WriteObject(stream1, myData);
        result = Encoding.UTF8.GetString(stream1.ToArray());
        dataArray.Add(result);
    }
}
serializerArray.WriteObject(stream, dataArray);
result = Encoding.UTF8.GetString(stream.ToArray());
this.WriteReturnStr(result);
}
}
private void ReadData()
{
    serializer = new DataContractJsonSerializer(typeof(Data));
    Stream instream = Request.InputStream;
    byte[] buffer = new byte[instream.Length];
    instream.Read(buffer, 0, buffer.Length);
    using (MemoryStream stream = new MemoryStream(buffer))
    {
        myData = serializer.ReadObject(stream) as Data;
    }
}
private bool OpenDBConnection()
{
    try
    {
        this.SqlCon.Open();
        return true;
    }
    catch (Exception e)
    {
        return false;
    }
}
public System.Data.DataSet GetDataFromDB(string SqlStr)
{
    try
    {
        DataSet ds = new DataSet();
        this.DataAdapter = new SqlDataAdapter(SqlStr, this.SqlCon);
        this.DataAdapter.Fill(ds);
        return ds;
    }
    catch (Exception e)

```

```

        {
            return null;
        }
    }
    private bool ExecSingleSql(string SqlStr)
    {
        try
        {
            this.Command = new SqlCommand(SqlStr);
            this.Command.Connection = this.SqlCon;
            this.Command.ExecuteNonQuery();
            return true;
        }
        catch (Exception e)
        {
            return false;
        }
    }
    private void WriteReturnStr(String ReturnStr)
    {
        Response.Write(ReturnStr);
        Response.Flush();
    }
}
}

```

先调用 ASP.NET 服务器端的脚本代码连接数据库，连接失败时将“数据库连接失败”文字返回给客户端。连接成功后读取客户端的传入参数 operateType 的参数值，并根据该参数值进行如下处理。

□ 传入参数 operateType 的参数值为“load”

如果传入参数 operateType 的参数值为“load”，则调用 LoadAllData 函数读取数据库中全部订单信息并将这些订单信息经过序列化处理后转换为字符串并返回给客户端。

在 LoadAllData 函数中，首先在数据库中获取全部订单信息，如果读取失败则将“读取数据失败”文字返回给客户端；如果读取成功，则利用 DataContractJsonSerializer 类将读取到的每一行数据进行序列化处理（在客户端可以通过 JSON 对象的 parse 方法将其解析为一个 JavaScript 对象，该对象的每个属性值为此处设置的每个订单数据的各个字段值），然后将序列化后的结果转换成为字符串放在一个数组中，最后使用 DataContractJsonSerializer 类将这个装有所有订单数据的数组进行序列化处理（在客户端可以通过 JSON 对象的 parse 方法将其解析成为一个由 JavaScript 对象组成的数组），然后将序列化后的结果转换成为字符串并返回给客户端。

□ 传入参数 operateType 的参数值为“add”

如果传入参数 operateType 的参数值为“add”，则首先调用 ReadData 函数将客户端提交的 data 对象转换成为服务器端可以使用的 Data 对象（Data 对象的各属性值中保存了表单上

各控件中的输入内容), 并将该 Data 对象赋值给变量 myData。然后在数据库中查找是否已经存在订单编号等于 myData 对象的 Code 值的订单信息, 如果已经存在, 则将“输入的订单编号在数据库中已存在”文字返回给客户端; 如果不存在, 则将 myData 对象中保存的订单信息追加到数据库中。如果追加数据失败, 则将“数据添加失败”文字返回给客户端; 如果添加成功, 则重新调用 LoadAllData 函数读取数据库中全部订单信息, 并且将这些订单信息经过序列化处理后转换为字符串并返回给客户端。

□ 传入参数 operateType 的参数值为“update”

如果传入参数 operateType 的参数值为“update”, 则首先调用 ReadData 函数将客户端提交的 data 对象转换成为服务器端可以使用的 Data 对象(Data 对象的各属性值中保存了表单上各控件中的输入内容), 并将该 Data 对象赋值给变量 myData。然后将 myData 对象中保存的订单信息更新到数据库中, 如果更新数据失败, 则将“数据修改失败”文字返回给客户端; 如果更新成功, 则重新调用 LoadAllData 函数读取数据库中全部订单信息, 并且将这些订单信息经过序列化处理后转换为字符串并返回给客户端。

□ 传入参数 operateType 的参数值为“delete”

如果传入参数 operateType 的参数值为“delete”, 则将数据库中订单编号等于传入的订单编号的订单信息删除。如果删除数据失败, 则将“数据删除失败”文字返回给客户端; 如果删除成功, 则重新调用 LoadAllData 函数读取数据库中全部订单信息, 并且将这些订单信息经过序列化处理后转换为字符串并返回给客户端。

## 8.2 案例 26: 在后台线程中实现数据的批量插入

### 8.2.1 案例概述

本节将讨论如何在后台线程中将一批数据插入到服务器端的数据库中。本案例的页面与实现功能和第 5 章案例 19 的页面与实现功能完全相同, 用户单击页面上的提交按钮后, 将保存在本地数据库中的订单数据追加到服务器端的数据库中, 不同之处在于本案例把订单数据追加到服务器端数据库中的操作在后台线程中执行, 因此当本地数据库中的订单数据比较多时用户可以在单击提交按钮之后立即执行页面上的下一步操作, 例如可以在表单中继续输入一条新的数据, 而不必等到本地数据库中的订单数据被追加到服务器端数据库中之后才执行下一步操作。

### 8.2.2 代码剖析

#### 1. 页面的 JavaScript 脚本代码

本案例页面的显示效果、HTML 页面代码与样式代码与第 5 章中案例 19 的页面显示效果、HTML 页面代码与样式代码完全相同, 故不再赘述。JavaScript 脚本代码也与第 5 章案

例 19 的 JavaScript 脚本代码大致相同, 唯一不同之处在于, 用户单击提交按钮之后所调用的函数代码与案例 19 中用户单击提交按钮之后所调用的函数代码完全不同, 因此这里只分析本案例中用户单击提交按钮后所调用的函数, 代码如下 (函数名为 btnSubmit\_onclick)。

```
function btnSubmit_onclick() {
    datatable = document.getElementById("datatable");
    var tr;
    var dataArray = new Array();
    for (var i = 2; i < datatable.childNodes.length; i++) {
        data = new Object();
        tr = datatable.childNodes[i];
        data.Code = tr.childNodes[0].innerHTML;
        data.Date = tr.cells[1].innerHTML;
        data.GoodsCode = tr.cells[2].innerHTML;
        data.BrandName = tr.cells[3].innerHTML;
        data.Num = tr.cells[4].innerHTML;
        data.Price = tr.cells[5].innerHTML;
        data.PersonName = tr.cells[7].innerHTML;
        data.Email = tr.cells[8].innerHTML;
        dataArray.push(JSON.stringify(data));
    }
    var worker = new Worker("代码清单 8-3Worker.js");
    worker.postMessage(JSON.stringify(dataArray));
    worker.onmessage = function (event) {
        datatable = document.getElementById("datatable");
        if (event.data == "数据库连接失败" || event.data == "提交数据失败")
            alert(event.data);
        else
        {
            db.transaction(function (tx) {
                tx.executeSql('delete from orders', [],
                    function (tx, rs) {
                        alert("成功提交数据, 本地数据库中数据被清空!");
                    },
                    function (tx, error) {
                        alert(error.source + "::-" + error.message);
                    }
                );
            });
            removeAllData();
        }
    }
}
```

btnSubmit\_onclick 函数首先获取页面上的订单信息一览表元素, 并把它赋值给全局变量 datatable, 然后利用变量 dataArray 定义一个 JavaScript 数组。接下来, 循环获取订单信息一览表中的所有数据 (这时该一览表中显示本地数据库中的所有订单数据), 在循环中利用变量 data 定义一个 JavaScript 对象, 并将一览表中每一行的各列数据保存在 data 对象的各个属性中, 然后使用 JSON 对象的 stringify 方法将 data 对象转换成为字符串并保存在



dataArray 数组中，循环结束后再通过 JSON 对象的 stringify 方法将 dataArray 数组转换为字符串。接下来将该字符串作为消息发送给定义后台线程中执行代码的脚本文件“代码清单 8-3Worker.js”，在后台线程中将利用 AJAX 调用 ASP.NET 服务器端脚本代码将 dataArray 数组中保存的所有订单数据追加到服务器端数据库中，并将是否追加成功的信息返回给客户端，后台线程接收到服务器端返回的消息后再传回给前台 JavaScript 脚本代码。前台 JavaScript 脚本代码在接收到这串消息后进行判断，如果消息字符串为“数据库连接失败”或“提交数据失败”，则直接将这个字符串显示在弹出提示消息窗口中；否则将本地数据库中所有订单数据删除。删除失败时弹出系统错误信息，删除成功时弹出提示信息，文字为“成功提交数据，本地数据库中数据被清空！”，并调用 removeAllData 函数将页面上的订单信息一览表中的订单信息全部清除。

## 2. 在后台线程中运行的 JavaScript 脚本代码

接下来看一下本案例中使用的定义后台线程中运行的 JavaScript 脚本代码，如代码清单 8-5 所示，代码文件名为“代码清单 8-3Worker.js”。

代码清单 8-5 后台线程中运行的 JavaScript 脚本代码

---

```
onmessage = function (event) {
    var xhr = new XMLHttpRequest();
    xhr.open("POST", "代码清单 8-3BackCode.aspx");
    xhr.onreadystatechange = function () {
        var result = xhr.responseText;
        if (xhr.readyState == 4) {
            postMessage(result);
        }
    }
    xhr.send(event.data);
}
```

---

在后台线程中运行的 JavaScript 脚本代码中，在接收到前台脚本代码发送的消息字符串后利用 AJAX 调用 ASP.NET 服务器端脚本代码并将接收到的消息字符串提交给该服务器端脚本代码，在接收到服务器端返回的消息字符串后将该消息字符串发送回前台 JavaScript 脚本代码中。

## 3. ASP.NET 服务器端代码

接下来看一下本案例中使用的 ASP.NET 服务器端脚本代码，如代码清单 8-6 所示，该服务器端脚本代码的文件名为“代码清单 8-3BackCode.aspx”。

代码清单 8-6 本案例中使用的 ASP.NET 服务器端脚本代码

---

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
```

---

```

using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Configuration;
using System.Data;
using System.Data.SqlClient;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Json;
using System.IO;
using System.Text;

namespace HTML5TEST
{
    public partial class 代码清单8_2BackCode : System.Web.UI.Page
    {
        [DataContract]
        public class Data
        {
            [DataMember]
            public String Code { get; set; }
            [DataMember]
            public String Date { get; set; }
            [DataMember]
            public String GoodsCode { get; set; }
            [DataMember]
            public String BrandName { get; set; }
            [DataMember]
            public String Num { get; set; }
            [DataMember]
            public String Price { get; set; }
            [DataMember]
            public String PersonName { get; set; }
            [DataMember]
            public String Email { get; set; }
        }

        private System.Data.SqlClient.SqlConnection SqlCon;
        private System.Data.SqlClient.SqlDataAdapter DataAdapter;
        private System.Data.SqlClient.SqlCommand Command;//SQL Command
        protected void Page_Load(object sender, EventArgs e)
        {
            string Constr, SqlStr, str;
            bool SuccessFlag;
            String strResult = String.Empty;
            Constr = System.Configuration.ConfigurationManager.
                ConnectionStrings["dbConnection"].ToString();
            this.SqlCon = new System.Data.SqlClient.SqlConnection();
            this.SqlCon.ConnectionString = Constr;
            SuccessFlag = OpenDBConnection();
            if (!SuccessFlag)
            {

```

```

        strResult = "数据库连接失败";
        this.WriteReturnStr(strResult);
        return;
    }
    Stream instream = Request.InputStream;
    byte[] buffer = new byte[instream.Length];
    instream.Read(buffer, 0, buffer.Length);
    DataContractJsonSerializer serializerArray =
        new DataContractJsonSerializer(typeof(ArrayList));
    DataContractJsonSerializer serializer =
        new DataContractJsonSerializer(typeof(Data));
    using (MemoryStream stream = new MemoryStream(buffer))
    {
        ArrayList dataArray = serializerArray.ReadObject(stream) as
            ArrayList;
        for (int i = 0; i < dataArray.Count; i++)
        {
            str = dataArray[i].ToString();
            buffer = Encoding.UTF8.GetBytes(str);
            using (MemoryStream stream1 = new MemoryStream(buffer))
            {
                Data myData = serializer.ReadObject(stream1) as Data;
                SqlStr = "insert into orders ";
                SqlStr += "values('" +
                    myData.Code.Trim().Replace("'", "'') + "','" +
                    myData.Date.Trim() + "','" +
                    myData.GoodsCode.Trim().Replace("'", "'') + "','" +
                    myData.BrandName.Trim().Replace("'", "'') + "','" +
                    myData.Num.ToString() + "','" +
                    myData.Price.ToString() + "','" +
                    myData.PersonName.Trim().Replace("'", "'') + "','" +
                    myData.Email.Trim().Replace("'", "'') + '');"
                SqlStr += myData.PersonName.Trim().Replace("'", "'') + "','" +
                    myData.Email.Trim().Replace("'", "'') + '');"
                SuccessFlag = this.ExecSingleSql(SqlStr);
                if (SuccessFlag == false)
                {
                    strResult = "提交数据失败";
                    this.WriteReturnStr(strResult);
                    return;
                }
            }
        }
    }
    strResult = "提交数据成功";
    this.WriteReturnStr(strResult);
    return;

```

```

    }
    private bool OpenDBConnection()
    {
        try
        {
            this.SqlCon.Open();
            return true;
        }
        catch (Exception e)
        {
            return false;
        }
    }
    private bool ExecSingleSql(string SqlStr)
    {
        try
        {
            this.Command = new SqlCommand(SqlStr);
            this.Command.Connection = this.SqlCon;
            this.Command.ExecuteNonQuery();
            return true;
        }
        catch (Exception e)
        {
            return false;
        }
    }
    private void WriteReturnStr(String ReturnStr)
    {
        Response.Write(ReturnStr);
        Response.Flush();
    }
}
}

```

当服务器端脚本代码被调用的时候，首先连接数据库，如果连接数据库失败，则将“数据库连接失败”字符串返回给客户端。连接数据库成功后将客户端提交的由 JavaScript 对象组成的数组解析成服务器端 Data 对象所构成的数组，然后对该数组进行遍历，将数组中每一个客户端的 JavaScript 对象解析成服务器端所使用的 Data 对象，并将该对象中的数据插入到 SQL Server 数据库中。如果插入失败，则将“提交数据失败”文字返回给客户端；如果插入成功，则将“提交数据成功”文字返回给客户端。

## 8.3 本章小结

本章通过两个案例详细阐述了 HTML 5 中 Web Workers 的概念及使用方法。希望读者通过学习本章内容，能够了解如何在网页中创建多线程，并且在线程代码中指定所需执行的处

理。最后来系统地看一下所有在线程的 JavaScript 脚本文件中可用的变量、函数与类（关于 Web Workers 的更多内容，可以参阅笔者所著的《HTML 5 与 CSS 3 权威指南》一书）。

#### ❑ self

self 关键词用来表示本线程范围内的作用域。

#### ❑ postMessage(message)

向创建线程的源窗口发送消息。

#### ❑ onmessage

获取接收消息的事件句柄。

#### ❑ importScripts(urls)

导入其他 JavaScript 脚本文件。参数为该脚本文件的 url 地址，可以导入多个脚本文件，如下所示。

```
importScripts('script1.js','scripts\script2.js',' scripts\script3.js');
```

导入的脚本文件必须与使用该线程文件的页面在同一个域中，在同一个端口中。

#### ❑ navigator 对象

与 window.navigator 对象类似，具有 appName,platform,userAgent,appVersion 等属性。

#### ❑ sessionStorage/localStorage

可以在线程中使用 Web Storage。

#### ❑ XMLHttpRequest

可以在线程中处理 Ajax 请求。

#### ❑ Web Workers

可以在线程中嵌套线程。

#### ❑ setTimeout()/setInterval()

可以在线程中实现定时处理。

#### ❑ close

可以结束本线程。

#### ❑ eval () ,isNaN () ,escape () 等

可以使用所有 JavaScript 核心函数。

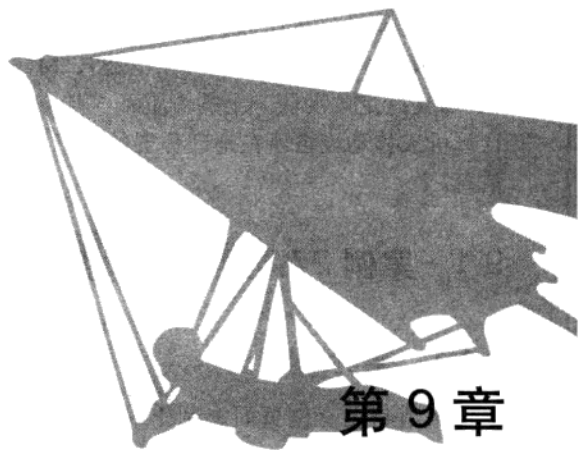
#### ❑ object

可以创建和使用本地对象。

#### ❑ WebSockets

可以在后台 JavaScript 脚本代码中使用 WebSockets 通信功能，并将接受到的消息或数据发送给前台页面的 JavaScript 脚本代码中进行后续处理。

下一章将通过一个比较典型的案例来阐述 HTML 5 中 Geolocation API 的基本知识及使用方法。读者可以通过第 9 章内容了解到，在 HTML 5 的网页中，如何通过使用 Geolocation API 来获取客户端计算机或移动设备的相关地理位置信息。



## 第 9 章

# 利用 Geolocation API 获取地理位置信息

### 本章内容

- 案例 27：显示计算机或移动设备所在地的地图
- 本章小结

本章通过一个比较典型的案例来详细阐述 HTML 5 中有关 Geolocation API 的基本知识及使用方法。Geolocation API 也是 HTML 5 中新增的一个重要 API。如果计算机或移动设备（必须支持定位功能）上的浏览器支持 Geolocation API，那么该浏览器中可以直接显示用户计算机或移动设备所在地理位置的信息，如地图、天气预报、地方新闻等与地理位置相关的信息。

## 9.1 案例 27：显示计算机或移动设备所在地的地图

### 9.1.1 案例概述

本节将在案例页面上制作一幅地图，并在页面中显示用户计算机或移动设备所在地的地图。

### 9.1.2 页面显示效果

首先来看一下案例页面在浏览器中的显示效果。

在浏览器中打开案例页面时，浏览器会询问用户是否共享用户计算机或移动设备的地理位置信息，如图 9-1 所示。



图 9-1 浏览器询问用户是否共享计算机或移动设备的地理位置信息

在不支持 Geolocation API 的浏览器中，打开浏览器时会显示错误提示信息，如图 9-2 所示。

在支持 Geolocation API 的浏览器中，当浏览器询问用户是否共享用户计算机或移动设备的地理位置信息时，选择共享地理位置信息，浏览器中将会显示用户计算机或移动设备所在地的地图，如图 9-3 所示。

在该页面中，用户单击监视位置更改按钮后，浏览器将会对用户计算机或移动设备所在地进行监视，每隔一段时间检查用户计算机或移动设备的地理位置是否发生改变。如果用户计算机或移动设备的地理位置发生改变则更新页面中的地图。用户单击停止监视按钮后该监

视被取消。

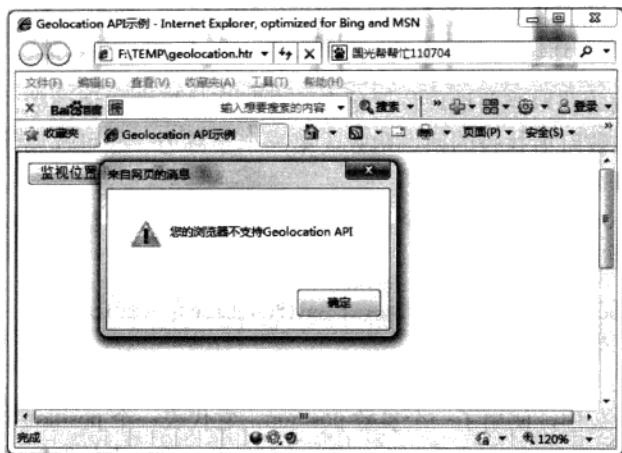


图 9-2 在不支持 Geolocation API 的浏览器中打开案例页面时会弹出错误提示信息

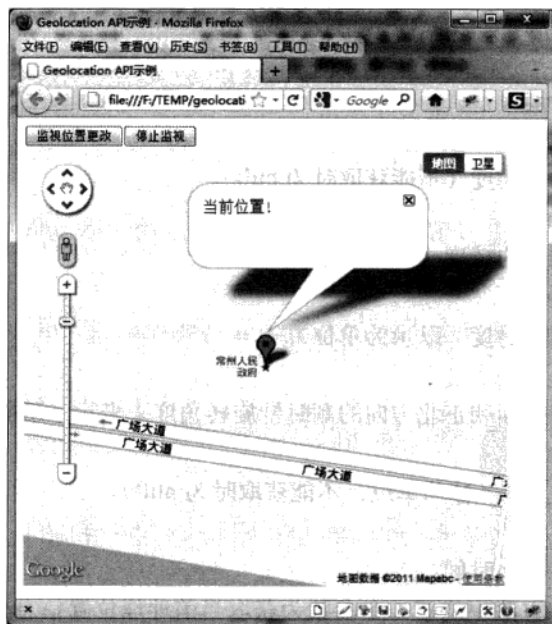


图 9-3 用户同意共享地理位置信息后浏览器中显示计算机或移动设备所在地的地图

### 9.1.3 案例知识点

接下来对本案例中使用的 HTML 5 中的 Geolocation API 进行必要的介绍, 关于 Geolocation API 的完整知识, 可以参阅笔者所著的《HTML 5 与 CSS 3 权威指南》一书。



首先，在 Geolocation API 中，使用 `getCurrentPosition` 方法来获取用户当前的地理位置信息，该方法的定义如下所示。

```
void getCurrentPosition(onSuccess, onError, options);
```

其中第一个参数为获取当前地理位置信息成功时所执行的回调函数，第二个参数为获取当前地理位置信息失败时所执行的回调函数，第三个参数为一些可选属性的列表。其中，第二和第三个参数为可选属性。

`getCurrentPosition` 方法中的第一个参数为获取当前地理位置信息成功时所执行的回调函数。该参数的使用方法如下所示。

```
navigator.geolocation.getCurrentPosition(function(position) {
    // 获取成功时的处理
})
```

在上面的回调函数中，使用了一个参数 `position`，它代表一个 `position` 对象，它的各个属性中保存的是获取到的地理位置信息。`position` 的具体属性及其中保存的地理位置信息如下所示。

❑ `latitude`

当前地理位置的纬度。

❑ `longitude`

当前地理位置的经度。

❑ `altitude`

当前地理位置的海拔高度（不能获取时为 `null`）。

❑ `accuracy`

获取到的纬度或经度的精度（以米为单位）。

❑ `altitudeAccuracy`

获取到的海拔高度的精度（以米为单位）。

❑ `heading`

设备的前进方向。用面朝正北方向的顺时针旋转角度来表示（不能获取时为 `null`）。

❑ `speed`

设备的前进速度（以米 / 秒为单位，不能获取时为 `null`）。

❑ `timestamp`

获取地理位置信息时的时间。

`getCurrentPosition` 方法中的第二个参数为获取当前地理位置信息失败时所执行的回调函数。如果获取地理位置信息失败，则可以通过该回调函数把错误信息提示给用户。

如果在浏览器中拒绝了共享位置信息，也会引起错误的发生。

该回调函数以一个 `error` 对象作为参数，该对象具有以下两个属性。

1) `code` 属性。

`code` 属性为以下三个值中的一个：

- ❑ 用户拒绝了位置服务（属性值为1）
- ❑ 获取不到位置信息（属性值为2）
- ❑ 获取位置信息超时错误（属性值为3）

## 2) message 属性。

message 属性为一个字符串。该字符串包含了错误信息，这个错误信息在开发和调试时很有用。有些浏览器，如 Firefox 3.6 以上版本，不支持 message 属性。

getCurrentPosition 方法中的第三个参数可以省略。该参数是一些可选属性的列表，这些可选属性如下。

### ❑ enableHighAccuracy

是否要求高精度的地理位置信息。这个参数在很多设备上不起作用，因为能否获得高精度的地理位置信息需要结合设备电量和具体地理情况来考虑。因此，一般把该属性设为默认，由设备自身来调整。

### ❑ timeout

对地理位置信息的获取操作设定一个超时限制（单位为毫秒）。如果在该时间内未获取到地理位置信息，则返回错误。

### ❑ maximumAge

对地理位置信息进行缓存的有效时间（单位为毫秒），例如，设定 maximumAge 为 120 000（1 分钟是 60 000 毫秒）。如果 10:00 时获取过一次地理位置信息，10:01 时再次调用 navigator.geolocation.getCurrentPosition 重新获取地理位置信息，则返回的依然为 10:00 时的数据（因为设置的缓存有效时间为 2 分钟）。超过这个时间后缓存的地理位置信息被废弃，尝试重新获取地理位置信息。如果该值被指定为 0，则无条件重新获取新的地理位置信息。

对于这些可选属性进行具体设置的方法如下所示。

```
navigator.geolocation.getCurrentPosition(
    function(position) {
        // 获取地理位置信息成功时所做的处理
    },
    function(error) {
        // 获取地理位置信息失败时所做的处理
    },
    // 以下为可选属性
    {
        // 设置缓存有效时间为 2 分钟
        maximumAge: 60*1000*2,
        // 5 秒钟内未获取到地理位置信息则返回错误
        timeout: 5000
    }
);
```

在 Geolocation API 中，watchPosition 方法可以连续获取用户的当前地理位置信息，它会

定期地自动获取用户的当前地理位置信息。该方法的定义如下所示。

```
int watchCurrentPosition(onSuccess, onError, options);
```

该方法中三个参数的说明和使用方法与 `getCurrentPosition` 方法中参数的说明和使用方法相同。该方法返回一个数字，这个数字的使用方法与 JavaScript 脚本中 `setInterval` 方法的返回参数的使用方法相似，可以被 `clearWatch` 方法使用，停止对当前地理位置信息的监视。

使用 `clearWatch` 方法可以停止对用户当前地理位置信息的监视。该方法的定义如下所示。

```
void clearWatch(watchId);
```

该方法的参数为调用 `watchCurrentPosition` 方法监视地理位置信息时的返回参数。

### 9.1.4 代码剖析

案例页面的详细代码如代码清单 9-1 所示。

代码清单 9-1 案例页面的详细代码

---

```
<!DOCTYPE html>
<head>
<meta name="viewport" content="width=620" />
<title>Geolocation API 示例</title>
<script type="text/javascript">
var streetNumber,street,city,province,country;
var watchId;
function window_onload() {
    if(navigator.geolocation==null)
    {
        alert("您的浏览器不支持 Geolocation API");
        document.getElementById("watchPosition").disabled="disabled";
        document.getElementById("clearWatch").disabled="disabled";
    }
    else
        navigator.geolocation.getCurrentPosition(showMap,onError,
            {enableHighAccuracy:true,timeout:60000});
}
function watchPosition() {
    watchId=navigator.geolocation.watchPosition(showMap);
}
function clearWatch()
{
    navigator.geolocation.clearWatch(watchId);
}
function showMap(position)
{
    var coords = position.coords;
    var latlng = new google.maps.LatLng(coords.latitude, coords.longitude);
    var myOptions = {
        zoom: 18,
```

```

        center: latlng,
        mapTypeId: google.maps.MapTypeId.ROADMAP
    });
    var map1= new google.maps.Map(document.getElementById("map"),
    myOptions);
    var marker = new google.maps.Marker({
        position: latlng,
        map: map1
    });
    var infowindow = new google.maps.InfoWindow({
        content: "当前位置!"
    });
    infowindow.open(map1, marker);
}
function onError(error)
{
    var message = "";
    switch (error.code) {
        case error.PERMISSION_DENIED:
            message = "位置服务被拒绝";
            break;
        case error.POSITION_UNAVAILABLE:
            message = "未能获取到位置信息";
            break;
        case error.PERMISSION_DENIED_TIMEOUT:
            message = "在规定时间内未能获取到位置信息";
            break;
    }
    if (message == "")
    {
        var strErrorCode = error.code.toString();
        message = "由于不明原因, 未能获取到位置信息 (错误号: "+strErrorCode+").";
    }
    alert(message);
    document.getElementById("watchPosition").disabled="disabled";
    document.getElementById("clearWatch").disabled="disabled";
}
</script>
<script type="text/javascript"
src=http://maps.google.com/maps/api/js?sensor=false>
</script>
</head>
<body onload="window.onload()">
    <input type="button" id="watchPosition" value="监视位置更改"
    onclick="watchPosition()" />
    <input type="button" id="clearWatch" value="停止监视"
    onclick="clearWatch()" />
    <div id="map" style="width:500px; height:460px"></div>
    <div id="information" />
</body>

```

接下来分析一下案例页面中的 JavaScript 脚本代码。

#### □ window\_onload 函数

打开页面时调用 window\_onload 函数。在该函数中，如果浏览器不支持 Geolocation API，则显示错误提示信息，文字为“您的浏览器不支持 Geolocation API”，并将页面中的监视位置更改按钮与停止监视按钮设为无效状态。如果浏览器支持 Geolocation API，则调用 getCurrentPosition 方法来取得用户的当前地理位置信息，同时指定 showMap 函数为获取当前地理位置信息成功时所执行的回调函数，指定 onError 函数为获取当前地理位置信息失败时所执行的回调函数，指定浏览器获取高精度的地理位置信息，获取地理位置的超时限制为 60 000 毫秒。

#### □ showMap 函数

showMap 函数为浏览器获取用户当前地理位置信息成功时所执行的回调函数，该函数接收一个参数 position 对象，该对象的各个属性中保存了浏览器所获取到的用户当前地理位置的各种地理信息。

在 showMap 函数内部，首先获取用户所在地的经度与纬度并将其保存在 coords 对象中，然后利用 latlng 对象指定一个 Google 地图上的坐标点，同时指定该坐标点的横坐标和纵坐标为用户所在地的经度值和纬度值。代码如下所示。

```
var coords = position.coords;
var latlng = new google.maps.LatLng(coords.latitude, coords.longitude);
```

下面通过 myOptions 对象来指定地图的放大倍数为 18，地图中心点为用户所在地，地图类型为 Google 地图中的普通地图类型（非卫星地图），代码如下所示。

```
var myOptions = {
    zoom: 18,
    center: latlng,
    mapTypeId: google.maps.MapTypeId.ROADMAP
};
```

创建 Google 地图，并使其在页面中显示，代码如下所示。

```
var map1= new google.maps.Map(document.getElementById("map"), myOptions);
```

在地图上创建标记，代码如下所示。

```
var marker = new google.maps.Marker({
    position: latlng,
    map: map1
});
```

设置标注窗口并指定标注窗口中的注释文字，代码如下所示。

```
var infowindow = new google.maps.InfoWindow({
    content: "当前位置!"
});
```

最后，打开标注窗口，代码如下所示。

```
infoWindow.open(map1, marker);
```

#### ❑ onError 函数

onError 函数为浏览器获取用户的当前地理位置信息失败时所执行的回调函数，该函数接收一个参数 error 对象。该对象的 code 属性中保存了在 Geolocation API 中指定的当获取地理位置失败时所返回的错误号。

在该函数内部，根据错误号，弹出各种错误提示信息文字，同时将页面中的监视位置更改按钮与停止监视按钮设为无效状态。

#### ❑ watchPosition 函数

用户单击监视位置更改按钮时调用 watchPosition 函数。在该函数内部指定浏览器对用户的地理位置进行监视，每隔一段时间检查用户的地理位置是否发生了改变。如果用户的地理位置发生改变，则调用 showMap 函数重新显示页面上的 Google 地图。

#### ❑ clearWatch 函数

用户单击停止监视按钮时调用 clearWatch 函数。在该函数内部取消对用户地理位置的监视。

## 9.2 本章小结

本章通过一个比较有代表性的案例详细介绍了 HTML 5 中 Geolocation API 的各个知识点及使用方法。通过本章的阅读，希望读者能够比较全面地了解 Geolocation API 中的各个知识点并将其灵活运用在自己的 Web 网站或 Web 应用程序中。

从下一章开始，将会陆续介绍几个 HTML 5 的框架与插件。用户可以直接利用这些第三方提供的框架与插件来更加快速地开发出更加强大的 HTML 5 版的 Web 网站或 Web 应用程序。下一章将介绍以 HTML 5 中 Web Socket API 为基础开发出来的 jWebSocket 服务器及其 API。jWebSocket 服务器及其 API 是一个比较完整且强大的网络服务器端组件，对 HTML 5 中 Web Sockets API 进行了很好的扩展与实现。用户可以直接利用该组件来搭建 WebSocket 服务器与客户端并实现服务器与多个客户端之间的 Socket 通信。



## 第 10 章

# 使用 jWebSocket 框架开发 Socket 通信程序

### 本章内容

- ☐ 安装与运行 jWebSocket
- ☐ 创建第一个利用 jWebSocket 进行通信的 Web 页面
- ☐ 创建 jWebSocket 服务器端的侦听器
- ☐ jWebSocket 中的令牌
- ☐ jWebSocket 中服务器端的插件
- ☐ jWebSocket 中的通道
- ☐ 案例 28：使用 jWebSocket 服务器创建简单聊天室
- ☐ 本章小结

本章主要针对以 HTML 5 中的 WebSocket API 为基础开发出来的 jWebSocket 框架进行详细介绍。jWebSocket 框架是一个成熟的可以用来实现 Socket 通信的框架，可以直接使用它所提供的服务器插件及 API 来开发强大的实现 Socket 通信的 Web 应用程序。

## 10.1 安装与运行 jWebSocket

本节将介绍如何安装与设置 jWebSocket 服务器端和客户端的运行环境，以及如何正常运行 jWebSocket 服务器。在阅读本节内容之前，先从“<http://jwebsocket.org/downloads/downloads.htm>”这个网站上下载 jWebSocket 的安装包。

### 10.1.1 安装 jWebSocket 服务器

首先介绍如何在包括 Windows、Mac OS 和 Linux 在内的各种操作系统上安装 jWebSocket 服务器。在下载的安装包内应包括一个 jWebSocketServer-<版本号>.jar 文件，该文件中包括了所有运行 jWebSocket 服务器时必需的库文件，同时提供了预置的文件目录结构。jWebSocket 服务器可以很容易地通过命令行窗口来启动，不需要进行任何安装或特殊设置。

jWebSocket 服务器是基于纯 Java 技术建立起来的，因此在运行 jWebSocket 服务器时一定要确保已经安装了 Java Runtime Environment (JRE) 1.6 或者更高版本，并且设置好 JAVA\_HOME 环境变量并将其指向 JAVA 的安装路径。在 Windows 操作系统中，推荐在 PATH 环境变量中添加 java.exe 文件的所在路径，否则需要调整安装包内提供的启动 jWebSocket 服务器时所使用的批处理文件。另外，设置 JWEBSOCKET\_HOME 环境变量并将其指向 jWebSocket 的安装路径。

下面介绍安装 jWebSocket 服务器的步骤。

1) 下载 jWebSocket 服务器安装包 (jWebSocketServer-<版本号>.zip)。该压缩文件中包括 jWebSocketServer-<版本号>.jar 文件，所有运行 jWebSocket 服务器时所必需的库文件以及 jWebSocketServer-<版本号>.bat 批处理文件。

2) 解压安装包。解压后的路径中包括 jWebSocketServer-<版本号> 目录，该目录就是 jWebSocket 服务器的根目录，该目录下包括 4 个子目录，其中 conf 子目录下包含一个用于对 jWebSocket 服务器进行配置的 jWebSocket.xml 文件。Libs 子目录下包含 jWebSocketServer.jar 文件与所有运行 jWebSocket 服务器时所必需的库文件。利用后面将要介绍的插件或过滤器对 jWebSocket 进行扩展时所需要的 jar 文件也必须放在这个目录下。bin 目录下包含所有的 Windows 可执行文件、作为 Windows 服务被使用时的文件、启动 jWebSocket 服务器时所需要使用的批处理文件以及安装与卸载 Windows 的 32 位或 64 位的服务时所需要使用的文件。Logs 目录下包含作为日志来使用的 jWebSocket.log 日志文件。

3) 设置 JWEBSOCKET\_HOME 环境变量并将其指向 jWebSocket 的根目录——jWebSocketServer-<版本号> 目录。



在 Mac OS X 操作系统下设置环境变量的方法如下。

打开一个终端并输入 set 命令来显示所有环境变量。确保已经设置了 JAVA\_HOME 变量并将其指向 Java SDK 的安装目录。在 Mac OS X 10.6 操作系统中, 该目录通常为 /Library/Java/Home。要设置环境变量, 创建或打开 ~/.MacOSX/environment.plist 文件并在该文件中书写环境变量 (~ 表示用户文件夹, 如 ./users/alexanderschulze/.MacOSX/environment.plist 文件)。

4) 在 Windows 操作系统中, 运行 bin 目录下的 jWebSocketServer.bat 这个批处理文件。在 bin 目录中, 同时为 Mac OS X 操作系统提供了一个 jWebSocketServer.command 脚本文件, 为 ubuntu 操作系统提供了一个 jWebSocketServer.sh 文件。如果 PATH 环境变量中没有包括 java.exe 文件的所在路径, 需要手工修改 jWebSocketServer.bat 这个批处理文件以使其能够找到 java.exe 文件。

5) 如果想在所有操作系统中手动地采用统一方法来启动 jWebSocket 服务器, 在命令行中输入 “java -jar bin/jWebSocketServer-<version>.jar”, 如图 10-1 所示 (该运行结果因 jWebSocketServer 版本的不同而不同)。

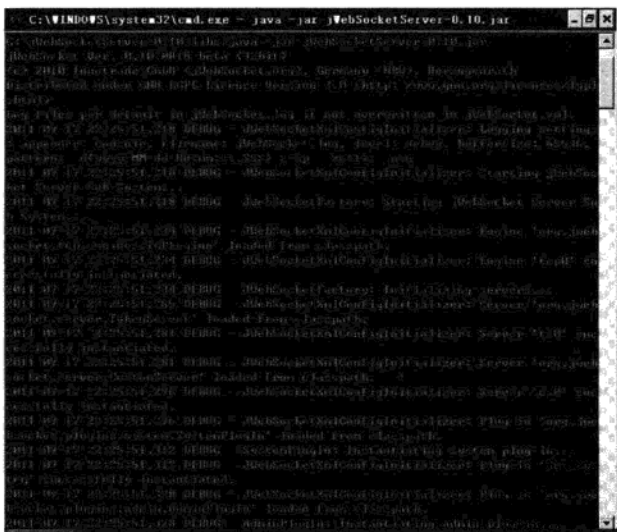


图 10-1 在命令行中运行 jWebSocketServer.jar 文件

在运行 jWebSocket 服务器时, 可以在命令行中添加一个 “-config <jWebSocket 服务器的配置文件的路径>” 参数, 可以在该参数中手动指定运行 jWebSocket 服务器时使用的配置文件及其路径, 而不使用默认的配置文。在为了测试目的而同时运行几个 jWebSocket 服务器并为每个服务器指定不同的配置文件时, 这个命令行参数是十分有用的。

### 10.1.2 在其他服务器环境下运行 jWebSocket 服务器

接下来介绍如何在运行其他服务器 (这里使用的是 Tomcat 服务器) 的环境中运行

jWebSocket 服务器，然后在同时运行这两个服务器的环境下运行 Web 应用程序。

为了与其他服务器相结合，jWebSocket 提供了一个 bundle 文件，该文件中包括所有必需的类库文件和 jWebSocket 默认使用的插件与过滤器。可以从“[http://jwebsocket.org/downloads/dl\\_bundles.htm](http://jwebsocket.org/downloads/dl_bundles.htm)”这个网站上下载这个 bundle 文件。为了学习如何结合使用 jWebSocket 服务器与 Tomcat 服务器，请确保当前使用的操作系统上已安装了 Tomcat 服务器（版本为 6.0 以上）与 Java 1.6（或以上版本）。接下来，将下载的 jWebSocketServer-Bundle-**<版本号>**.jar 文件放入 Tomcat 服务器的安装目录下的 lib 目录中。

在“<http://jwebsocket.org/downloads/downloads.htm>”中能够找到两个可用于部署的后缀名为 .war 的 Web 应用程序文件，这两个文件已通过 Tomcat 服务器的测试。第一个 Web 应用程序的文件中只包含一个用于首页文件的 index.htm 文件，如果已经熟悉 jWebSocket，并且想创建通过 jWebSocket 来进行 Socket 通信的 Web 应用程序，可以在这个应用程序的基础上继续扩展，使其成为自己的 Web 应用程序。第二个 Web 应用程序的文件中包含了一个完整的 jWebSocket 客户端应用程序，该客户端应用程序中包括各种用于演示的示例程序。这两个 Web 应用程序在运行时都需要使用前面提到过的 bundle 文件，运行这两个 Web 应用程序时需要执行如下步骤。

1) 从“<http://jwebsocket.org/downloads/downloads.htm>”这个网站上下载 jWebSocketAppServer-**<版本号>**.zip 文件或 jWebSocketAppSrvDemo-**<版本号>**.zip 文件。

2) 解压下载的 jWebSocketAppServer-**<版本号>**.zip 文件或 jWebSocketAppSrvDemo-**<版本号>**.zip 文件。

3) 启动 Tomcat 服务器，在“<http://localhost:<端口号>>”上以管理员身份进入 Tomcat 服务器的管理页面。

4) 部署后缀名为 .war 的 Web 应用程序文件，如图 10-2 所示。

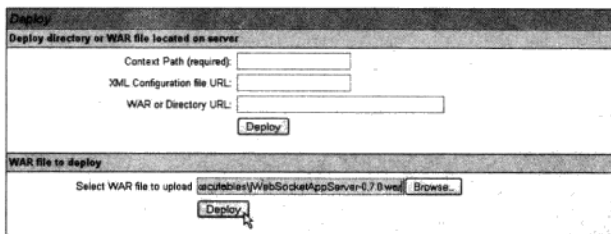


图 10-2 在 Tomcat 服务器上部署 jWebSocketAppServer 应用程序

5) 现在可以在 Tomcat 管理页面中停止并重新启动 Web 应用程序，如图 10-3 所示。

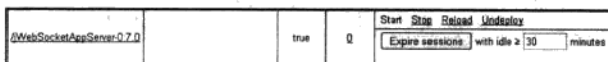


图 10-3 在 Tomcat 6.0 中停止并重新启动应用程序

6) 如果以自动部署 (auto-deploy) 的模式运行 Tomcat 服务器（可以在运行 Tomcat 服务器时使用的 server.xml 文件中查看是否以自动部署的模式运行 Tomcat 服务器），可以直接

将后缀名为 .war 的文件复制到 Tomcat 服务器所使用的 webapps 目录下。

7) 现在可以在浏览器中访问“http://localhost:<端口号>/jWebSocketAppServer-<版本号>”和“http://localhost:<端口号>/jWebSocketAppSrv-<版本号>”这两个 URL 地址。这两个地址在浏览器中分别显示默认的 index.htm 文件和有关演示程序的页面。

### 10.1.3 将 jWebSocket 服务器作为 Windows 的可执行文件

本小节描述如何将 jWebSocket 服务器作为 Windows 系统中的可执行文件来运行。如果想以 jWebSocket 服务器为基础来开发一些客户端应用程序（无论是 C/S 架构的还是 B/S 架构的），本小节中的内容会很有帮助。在可执行文件中内置了许多 jWebSocket 的 .jar 文件、程序图标及在安装 jWebSocket 时必须使用的类库文件。通过双击计算机中的 jWebSocket 的 Windows 可执行文件来轻松地运行 jWebSocket 服务器，无需进一步地安装或进行任何特殊的设置。

在运行 jWebSocket 的 Windows 可执行文件之前，首先要确保已经安装了一个可以正常运行的 jWebSocket 服务器。安装 jWebSocket 服务器的步骤请参阅 10.1.1 节。

在安装了 jWebSocket 服务器并确保其可以正常运行以后，可以通过以下步骤来运行 jWebSocket 在 Windows 系统中的可执行文件。

- 1) 根据需从“http://jwebsocket.org/downloads/downloads.htm”上下载 jWebSocketServer32-<版本号>.zip 文件或 jWebSocketServer64-<版本号>.zip 文件。
- 2) 解压下载的 .zip 文件，将 jWebSocketServer32/64-<版本号>.exe 文件复制到 jWebSocket 安装文件夹下的 bin 文件夹中（例如，c:\program files\jWebSocket-<版本号>\bin）。
- 3) 双击 jWebSocketServer32/64-<版本号>.exe 文件，将会立即运行 jWebSocket 服务器，如图 10-4 所示（该运行结果因 jWebSocketServer 版本的不同而不同）。

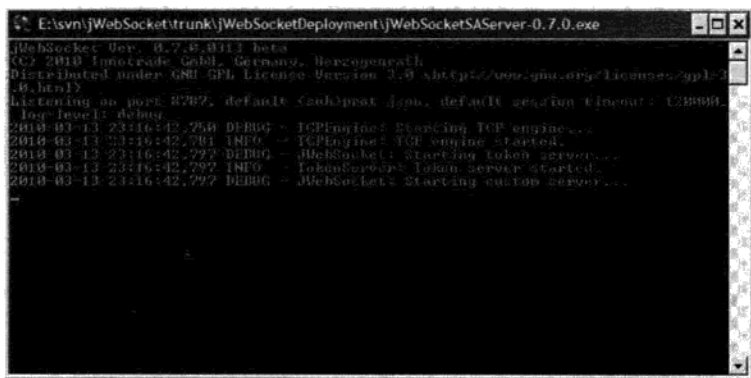


图 10-4 通过双击可执行文件来运行 jWebSocket 服务器

- 4) 在默认情况下，jWebSocket 将 jWebSocket 根目录（即 jWebSocket 的安装文件夹）下 /logs 文件夹中的 jWebSocket.log 文件作为 jWebSocket 服务器在运行时的日志文件。

### 10.1.4 将 jWebSocket 服务器作为 Windows 的服务

本小节将阐述如何安装 jWebSocket 服务器使其能够作为 Windows XP 或 Windows 7 等 Windows 操作系统中的服务来运行。如果已经开发好了基于 jWebSocket 服务器的客户端应用程序（无论是 B/S 架构的还是 C/S 架构的），在最终运行该应用程序的计算机中安装其运行环境时，本小节的内容会有所帮助。

在将 jWebSocket 服务器作为服务运行之前，要确保已经安装了一个可以正常运行的 jWebSocket 服务器。安装 jWebSocket 服务器的步骤请参阅 10.1.1 节。

可以通过以下步骤来安装或卸载运行 jWebSocket 服务器的服务。

1) 根据需从“<http://jwebsocket.org/downloads/downloads.htm>”上下载 jWebSocketService32-<版本号>.zip 文件或 jWebSocketService64-<版本号>.zip 文件。

2) 解压 .zip 文件，将 jWebSocketService32/64-<版本号>.exe 文件、jWebSocketInstallService32.bat 文件与 jWebSocketUninstallService32.bat 文件复制到 jWebSocket 安装文件夹下的 bin 文件夹中（例如，c:\program files\jWebSocket-<版本号>\bin）。

3) 在命令行窗口的 jWebSocket 安装文件夹下的 bin 文件夹中运行 jWebSocketInstallService32/64-<版本号>.bat 文件，将 jWebSocketService32/64-<版本号>.exe 可执行程序注册成为一个 Windows 的服务。

4) 在 Windows 的服务管理器中查看 jWebSocket 服务是否已被正确安装（通过“控制面板”中“管理工具”窗口下的“服务”管理窗口）。

5) 右击 jWebSocket 服务，并且选择“开始”选项。jWebSocket 服务开始运行，jWebSocket 服务器被启动。

6) 重启计算机后，jWebSocket 服务将自动运行。

7) 可以通过运行 jWebSocketUninstallService32/64-<版本号>.bat 文件来卸载 jWebSocket 服务。

8) 在默认情况下，jWebSocket 将 jWebSocket 根目录（即 jWebSocket 的安装文件夹）下的 /logs 文件夹中的 jWebSocket.log 文件作为 jWebSocket 服务器在运行时的日志文件。

### 10.1.5 jWebSocket Web 客户端

本小节将阐述如何使用 jWebSocket Web 客户端，并且运行 jWebSocket 演示程序。jWebSocket Web 客户端是基于纯 JavaScript 环境的。由于许多浏览器都提供了对 jWebSocket 框架所使用的核心 API——WebSocket API 的支持，所以在浏览器中不再需要安装任何插件。jWebSocket 框架的使用者或学习者甚至可以在本地文件系统中直接运行演示用 Web 网站而不需要安装任何 Web 服务器，目前 Firefox4.0 浏览器、Opera11.0 浏览器、Chrome10.0 浏览器和 Safari5.0 浏览器中都内置了对于 WebSocket API 的支持。

所有尚未支持 WebSocketAPI 的浏览器都内置了 FlashBridge 协议。对于应用程序来说，FlashBridge 协议是完全透明的，在开发应用程序的时候并不需要了解它的太多细节。

一旦浏览器对 WebSocketAPI 提供了支持, FlashBridge 协议将被自动关闭。但是, 在运行 FlashBridge 协议时需要 Adobe Flash 播放器插件与一个类似 Apache 的 Web 服务器, 因为 Adobe Flash 播放器并不支持文件协议。

在 Windows 操作系统中, 需要执行如下步骤来运行 jWebSocket 客户端。

- 1) 新建一个文件夹 (例如, C:\jWebSocket)。
- 2) 从 “<http://jwebsocket.org/downloads/downloads.htm>” 上下载 jWebSocketClient-<版本号>.zip 文件并将该文件解压缩到新建的文件夹中, 该压缩包中包含一个 jWebSocketClient-<版本号> 文件夹, 因此不需要手工创建这个文件夹。
- 3) 打开一个支持本地 WebSocket 的浏览器。
- 4) 在浏览器的地址栏中输入新建文件夹下 jWebSocketClient-<版本号> 文件夹中的 index.html 文件的 URL 地址 (例如, file:///c:/jWebSocket/jWebSocketClient-<版本号>/index.html)。
- 5) 浏览器中将打开 jWebSocket 客户端的欢迎页面, 如图 10-5 所示。接下来可以测试所有 jWebSocket 的演示程序 (使用 Google Chrome 浏览器)。



图 10-5 jWebSocket 客户端的欢迎页面

在运行 Apache 2.2 Web 服务器的环境中, 可以执行如下步骤来运行 jWebSocket 的客户端。

- 1) 新建一个文件夹 (例如, C:\jWebSocket)。
- 2) 从 “<http://jwebsocket.org/downloads/downloads.htm>” 上下载 jWebSocketClient-<版本号>.zip 文件并将该文件解压缩到新建的文件夹中, 该压缩包中包含一个 jWebSocketClient-<版

本号> 文件夹，因此不需要手工创建这个文件夹。

3) 在 Apache 服务器的 httpd.conf 文件的 <VirtualHost> 一节中添加如下代码（其中 jwc 代表了 jWebSocket Web 客户端）。

```
Alias /jwc "c:/jWebSocket/client"
<Directory "c:/jWebSocket/client" >
    Order allow,deny
    Allow from all
</Directory>
```

4) 同时确保在 httpd.conf 文件的 DirectoryIndex 一行中指定了 index.html 文件，代码如下所示。

```
<IfModule dir_module>
DirectoryIndex index.htm index.html index.jsp index.php
</IfModule>
```

5) 重启 Apache 服务器来应用新的配置。

6) 打开一个支持 WebSocket API 的浏览器。

7) 在地址栏中输入指向演示程序中 index.html 文件的 URL 地址（例如，http://localhost/jwc/index.html）。

8) 在浏览器中将打开 jWebSocket 客户端的欢迎页面。接下来可以测试所有 jWebSocket 的演示程序。

一定要打开防火墙中的 80 端口使计算机能够接收 http 请求，打开 8787 端口使计算机能够接收 WebSocket 请求。

## 10.2 创建第一个利用 jWebSocket 进行通信的 Web 页面

本节介绍一个最简单的利用 jWebSocket 进行通信的 Web 页面。该页面中使用的元素如表 10-1 所示。

表 10-1 页面中使用的元素

元素名称	元素类型	显示文字	说 明
“用户名:” 文字	页面文字	用户名:	
用户名文本框	input type="text"		用户在其中输入登录 jWebSocket 服务器时所使用的用户名
“密码:” 文字	页面文字	密码:	
密码文本框	input type="text"		用户在其中输入登录 jWebSocket 服务器时所使用的密码
“发送消息:” 文字	页面文字	发送消息:	
发送消息文本框	input type="text"		用户在其中输入需要发送给其他所有已经登录到 jWebSocket 服务器的用户的文本格式的消息

(续)

元素名称	元素类型	显示文字	说 明
建立连接按钮	input type="button"	建立连接	用户单击该按钮与 jWebSocket 服务器建立连接并登录。在 HTML 代码中将该按钮设定为无效状态, 页面被打开时如果浏览器支持 WebSocket 通信, 则该按钮为有效状态
广播消息按钮	input type="button"	广播消息	用户单击该按钮将发送消息文本框内的输入文字发送给其他所有登录到 jWebSocket 服务器的用户。页面被打开时该按钮为无效状态, 用户连接并登录到 jWebSocket 服务器时该按钮变为有效状态, 用户与 jWebSocket 服务器断开连接后该按钮变为无效状态
关闭连接按钮	input type="button"	关闭连接	用户单击该按钮从 jWebSocket 服务器中退出并显式关闭与 jWebSocket 服务器之间的连接。页面被打开时该按钮为无效状态, 用户连接并登录到 jWebSocket 服务器时该按钮变为有效状态, 用户与 jWebSocket 服务器断开连接后该按钮变为无效状态
通信消息显示区域	div		在该 div 中显示从 jWebSocket 服务器发送过来的消息

页面在浏览器中被打开时的显示效果如图 10-6 所示 (目前暂不将该页面放入任何 Web 工程中, 使用的浏览器为 Google Chrome 浏览器)。

如果浏览器不支持 WebSocket 通信功能, 则打开页面时的显示效果如图 10-7 所示。

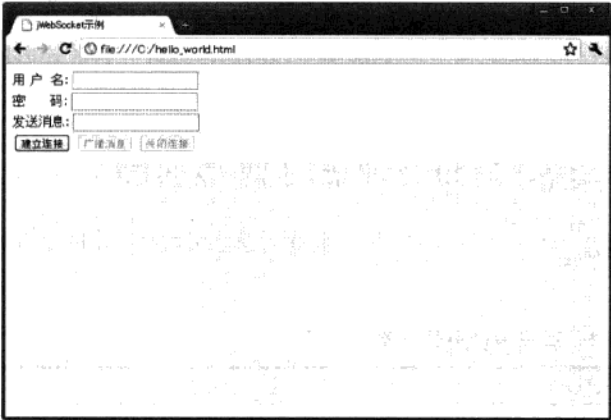


图 10-6 页面在浏览器中被打开时的显示效果

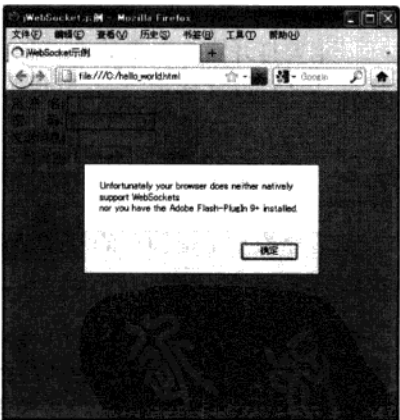


图 10-7 案例页面在不支持 WebSocket 通信的浏览器中被打开时的显示效果

用户输入用户名与密码后, 单击建立连接按钮与 jWebSocket 服务器建立连接, 登录到 jWebSocket 服务器。登录成功后通信消息显示区域中显示有关成功建立连接的消息, 如图 10-8 所示。

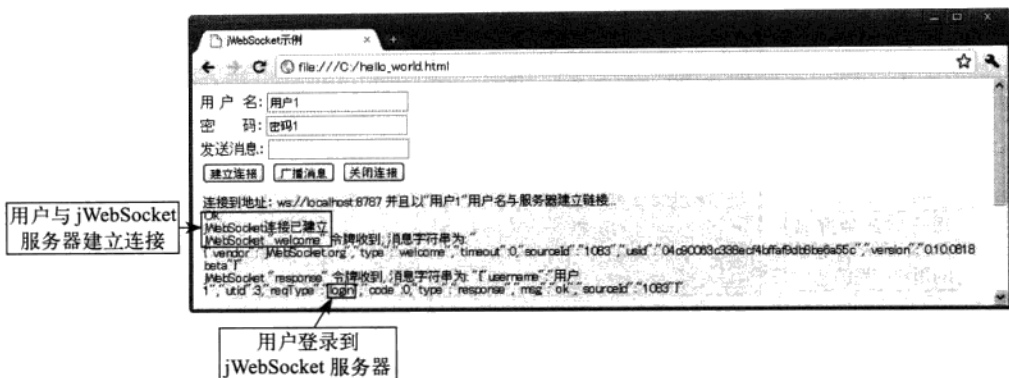


图 10-8 用户连接并登录到 jWebSocket 服务器

当其他用户连接并登录到 jWebSocket 服务器时，本页面中会显示该用户连接并成功登录到 jWebSocket 服务器的相关信息。

重新在一个 Google Chrome 浏览器中打开本页面，在该页面中通过另一组用户名和密码与 jWebSocket 服务器建立连接。本页面中显示该用户与 jWebSocket 服务器建立连接的消息，如图 10-9 所示（执行完这一步时暂且不要关闭本次打开的 Google Chrome 浏览器）。

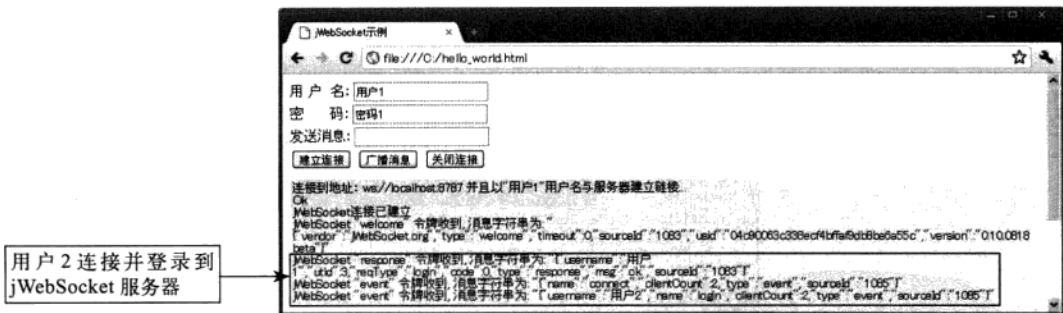


图 10-9 用户 2 连接并成功登录到 jWebSocket 服务器

在发送消息文本框中输入需要广播的消息并单击广播消息按钮，该消息会被 jWebSocket 服务器广播给其他所有连接并成功登录到 jWebSocket 服务器的用户，本页面中会显示与 jWebSocket 服务器进行通信的相关信息，如图 10-10 所示。

当其他用户向登录到 jWebSocket 服务器的所有用户广播消息时，本页面也会接收到该用户广播的消息。

在第二个被打开的 Google Chrome 浏览器中广播一条消息，然后查看本页面，页面中会显示接收到的该用户广播的消息，如图 10-11 所示。



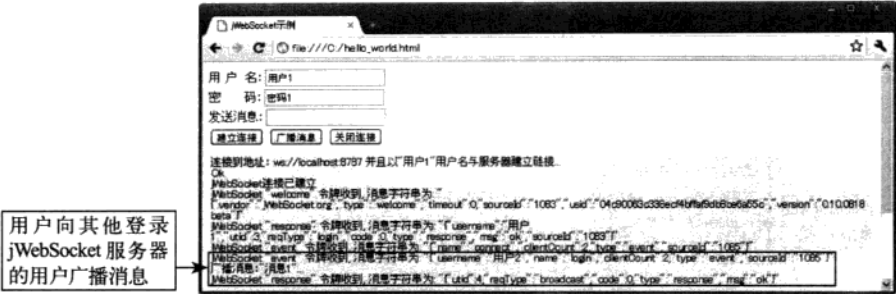


图 10-10 用户向其他用户广播消息

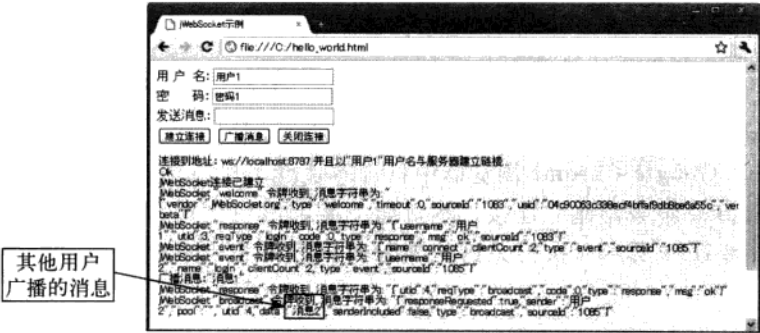


图 10-11 当其他用户广播消息时本页面会显示该用户广播的消息

当单击关闭连接按钮时，用户从 jWebSocket 服务器中退出并且显式关闭与 jWebSocket 之间的连接，如图 10-12 所示。



图 10-12 用户显式关闭与 jWebSocket 服务器的连接

接下来看一下本页面的完整代码，如代码清单 10-1 所示。

代码清单 10-1 本页面中的完整代码

```

<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>jWebSocket 示例 </title>
<style>
div#msg{
    border: 0px;
    margin:10px 0px 10px 0px;
    padding: 3px;
    background-color: #f0f0f0;
    -moz-border-radius: 5px;
    -webkit-border-radius: 5px;
    position:relative;
    height:300px;
    overflow:auto;
    font-size: 14px;
}
</style>
<script type="text/javascript" src="jWebSocket.js"></script>
<script type="text/javascript" language="JavaScript">
var jWebSocketClient;
var userName;
function window_onload()
{
    if( jws.browserSupportsWebSockets() ) {
        jWebSocketClient = new jws.jWebSocketJSONClient();
        document.getElementById("btnConnect").disabled="";
    }
    else {
        var lMsg = jws.MSG_WS_NOT_SUPPORTED;
        alert( lMsg );
    }
}
function btnConnect_click()
{
    var lURL = jws.JWS_SERVER_URL;
    userName = document.getElementById("userName").value;
    var userPass = document.getElementById("userPass").value;
    var msg=document.getElementById("msg");
    msg.innerHTML=" 连接到地址: " + lURL + " 并且以 \" " + userName +
    "\" 用户名与服务器建立连接 ... " ;
    var lRes = jWebSocketClient.logon(lURL,userName,userPass, {
        OnOpen: function( aEvent ) {
            msg.innerHTML+="<br/>jWebSocket 连接已建立 ";
        },
        OnMessage: function( aEvent, aToken ) {
            msg.innerHTML+="<br/>jWebSocket \" " +

```

```

        aToken.type + "\" 令牌收到，消息字符串为：\" + aEvent.data +
        "\"\" ;
    },
    OnClose: function( aEvent ) {
        msg.innerHTML+="  
>jWebSocket 连接被关闭. \" ;
        document.getElementById("btnbroadcastText").disabled="disabled";
        document.getElementById("btnDisConnect").disabled="disabled";
    }
});
msg.innerHTML+="  
>jWebSocketClient.resultToString(lRes);
if(lRes.code==0)
{
    document.getElementById("btnbroadcastText").disabled="";
    document.getElementById("btnDisConnect").disabled="";
}
}
function btnbroadcastText_click()
{
    var sendMsg=document.getElementById("sendMsg").value;
    var msg=document.getElementById("msg");
    msg.innerHTML+="  
>广播消息: \""+sendMsg+"\"...";
    var lRes = jWebSocketClient.broadcastText("",sendMsg);
    if(lRes.code!=0)
        msg.innerHTML=jWebSocketClient.resultToString( lRes );
    document.getElementById("sendMsg").value="";
}
function btnDisConnect_click()
{
    if(jWebSocketClient)
    {
        var msg=document.getElementById("msg");
        msg.innerHTML+="  
>用户 \"+"\""+userName+"\" 关闭连接 ";
        var lRes=jWebSocketClient.close();
        msg.innerHTML+="  
>jWebSocketClient.resultToString( lRes );
        if(lRes.code==0)
        {
            document.getElementById("btnbroadcastText").disabled="disabled";
            document.getElementById("btnDisConnect").disabled="disabled";
        }
    }
}
function window_onunload()
{
    if(jWebSocketClient)
    {
        jWebSocketClient.close({timeout:3000});
    }
}
</script>
<body onload="window_onload()" onunload="window_onunload()">

```



用户正在连接 jWebSocket 服务器，文字内容为：“连接到地址：” + jWebSocket 服务器所在地的 URL 地址 + “并且以” + 用户登录 jWebSocket 服务器时所使用的用户名 + “用户名与服务器建立连接……”。接下来使用 jWebSocketJSONClient 对象的 logon 方法建立连接并通过输入的用户名与密码登录到 jWebSocket 服务器，将登录结果赋值给变量 lRes，同时指定连接建立后所执行的回调函数，指定方法如下所示（aEvent 参数表示触发回调函数的事件）。

```
OnOpen: function( aEvent ) {
    // 回调函数中的代码
}
```

在该回调函数中指定在通信消息显示区域中显示“jWebSocket 连接已建立”文字。

指定当客户端接收到服务器端发送的令牌后所执行的回调函数，指定方法如下（aToken 参数表示服务器端发送给客户端的令牌，aToken.type 表示本次发送的令牌的类型，后面会针对令牌进行详细介绍）。

```
OnMessage: function( aEvent, aToken ) {
    // 回调函数中的代码
}
```

在回调函数中指定，当接收到服务器端发送的令牌时在通信消息显示区域中显示这样的文字：“jWebSocket” + 本次发送的令牌信息的类型 + “令牌收到，消息字符串为：” + aEvent 参数的 data 属性值（即服务器端发送的消息）。

指定当客户端与 jWebSocket 服务器端之间的连接被服务器端显式关闭时所执行的回调函数，指定方法如下所示。

```
OnClose: function( aEvent ) {
    // 回调函数中的代码
}
```

在回调函数中指定，当客户端与 jWebSocket 服务器端之间的连接被服务器端显式关闭时在通信消息显示区域中显示“jWebSocket 连接被关闭。”文字，将广播消息按钮与关闭连接按钮设为无效状态。

最后，btnConnect\_click 函数将连接与登录 jWebSocket 服务器端的登录结果（是否登录成功）显示在通信消息显示区域中。如果连接与登录 jWebSocket 服务器端操作成功，则将广播消息按钮与关闭连接按钮设为有效状态。

#### ❑ btnbroadcastText\_click 函数

用户在发送消息文本框中输入用户要发送给其他所有登录到 jWebSocket 服务器的用户的消息后，单击广播消息按钮会调用 btnbroadcastText\_click 函数。该函数首先获取用户在发送消息文本框中输入的内容并赋值给变量 sendMsg，获取页面上通信消息显示区域所使用的元素并赋值给变量 msg，然后在通信消息显示区域中显示文字：“广播消息：” + 用户在发送消息文本框中输入的内容。接下来使用 jWebSocketJSONClient 对象的 broadcastText 方法将用户在发送消息文本框中输入的内容发送给其他所有登录到 jWebSocket 服务器的用户。broadcastText 方法

有两个参数，其中第一个参数可以用来指定 socket 连接池，参数值为空时表示发送给其他所有登录到 jWebSocket 服务器的用户。第二个参数用来指定要发送的消息。发送消息后将发送结果赋值给变量 lRes。如果发送后变量 lRes 的 code 属性值不为 0，则表示发送失败，将发送结果显示在通信消息显示区域中。最后，将发送消息文本框中的内容清空。

这里需要注意的是，当使用 broadcastText 方法向其他用户发送消息时，发送过程是非阻塞型的，也就是说执行发送的线程不必等到接收方做出响应即可返回，但是发送方可以接收到接收方发出的异步响应。当接收到异步响应时执行如下回调函数中的代码。

```
OnMessage: function( aEvent, aToken ) {
    // 回调函数中的代码
}
```

#### ❑ btnDisconnect\_click 函数

用户单击关闭连接按钮时调用 btnDisconnect\_click 函数。该函数首先判断全局变量 jWebSocketClient 是否为 NULL，如果为 NULL，则表示浏览器不支持 WebSocket 通信，因此直接退出函数；如果 jWebSocketClient 全局变量不为 NULL，则获取页面上的通信消息显示区域所使用的元素并赋值给变量 msg，然后在通信消息显示区域中显示文字：“用户”+ 用户登录 jWebSocket 服务器时所使用的用户名 + “关闭连接”，之后利用 jWebSocketJSONClient 对象的 close 方法显式关闭与 jWebSocket 服务器之间的连接，并将关闭结果显示在通信消息显示区域中。如果关闭成功，则将广播消息按钮与关闭连接按钮设为无效状态。

在 close 方法中，可以指定一个 timeout 参数。该参数用来指定一个超时时间，以确保客户端在接收到服务器端的响应令牌后再关闭连接，比如将 timeout 参数值设定为 3000（毫秒），客户端将为服务器端的响应令牌的接收等待最多 3 秒，在接收到服务器端的响应令牌之后再关闭连接。如果不指定该参数，则客户端立即关闭与服务器之间的连接。

#### ❑ window\_onunload 函数

关闭页面时调用 window\_onunload 函数。该函数首先判断全局变量 jWebSocketClient 是否为 NULL，如果为 NULL 则表示浏览器不支持 WebSocket 通信，因此直接退出函数；如果全局变量 jWebSocketClient 不为 NULL，则表示客户端与 jWebSocket 服务器之间已建立连接，这时使用 jWebSocketJSONClient 对象的 close 方法显式关闭连接，并且指定 timeout 参数值为 3000 毫秒。

## 10.3 创建 jWebSocket 服务器端的侦听器

本节将介绍如何创建一个 jWebSocket 服务器端的侦听器。在创建侦听器之前，首先介绍 jWebSocket 服务器层的通信架构。

### 10.3.1 jWebSocket 的通信架构

WebSocket 协议是一种基于 TCP 套接字（Socket）通信协议之上的通信协议，但是当使

用 WebSocket 协议的时候，浏览器与客户端只需要做一个握手动作，浏览器和服务器之间就形成了一条快速通道，两者之间可以直接互相传送数据。在使用 HTTP 协议进行通信的时候，浏览器需要不断向服务器发出请求，由于 HTTP request 的 header 非常长，而其中包含的数据可能是一个很小的值，这样会占用很多的带宽和服务器资源。使用 Socket 协议进行通信就避免了这种情况，可以很好地节省服务器资源和带宽，同时也实现了实时通信。一个服务器可以处理几百个并发的客户端连接。由于 jWebSocket 的可扩展性，使用多个服务器组成的群集几乎可以支持无限个客户端（这也是 jWebSocket 2.0 版的实现目标）。

在使用 jWebSocket 服务器时，每个客户端与 jWebSocket 服务器的一个连接器（connector）建立连接。连接器由一个类似 jWebSocket 的内部 TCP 引擎（Engine）或诸如 Jbos Netty 之类的第三方引擎所驱动（jWebSocket 的核心服务层可以驱动很多第三方引擎）。图 10-13 展示了 jWebSocket 的通信架构（后面将对其中的有关名词及概念进行具体讲解）。

System 插件	用户 插件	用户 侦听器 1	用户 侦听器 2	自定义 插件 1	自定义 插件 2	自定义 侦听器	自定义应用程序 (不使用插件)	
令牌插件链		令牌侦听器链		自定义 插件链		侦听器链		
令牌服务器				自定义服务器				
JSON 处理器	CSV 处理器	XML 处理器	自定义 处理器	自定义数据包处理器				
服务器								
TCP 引擎		Netty 引擎		MINA 引擎		Jetty 引擎		
TCP 连接器	TCP 连接器	Netty 连接器	Netty 连接器	MINA 连接器		MINA 连接器	Jetty 连接器	Jetty 连接器

图 10-13 jWebSocket 的通信架构

### 10.3.2 创建侦听器

接下来看一下如何创建一个 jWebSocket 服务器端的侦听器（Listener）。通过创建侦听器，开发者可以更加容易地处理服务器端接收到的消息，可以自定义客户端与服务器端建立连接、关闭连接或执行其他操作时服务器端所执行的处理。

与插件（plug-in, 将在后面的内容中进行介绍）不同的是，侦听器被内置在开发者的应用程序的代码内，而且与应用程序的代码是紧密联系的。因此，侦听器通常被用来实现应用程序内的某些特定逻辑，而不是用来指定应用程序内的通用功能。当然，可以在侦听器内随意定义任何满足需要的逻辑，也可以在多个应用程序内部共享这些逻辑。与插件不同的是，使用侦听器的好处是只需实现一个接口就可以直接将它内置在代码内部。

只需经过如下几个步骤，就可以在自己的网站中创建侦听器并实现自定义逻辑。

### 1. 创建一个服务器端的侦听器类

要实现自定义侦听器类，第一步就是创建一个继承 WebSocketListener 类或 WebSocketTokenListener 类的 jWebSocket 侦听器类。在代码清单 10-2（代码清单 10-2 的文件名为 JWebSocketTokenListenerSample.java，包名为 serverTest）中自定义一个当客户端发送 getInfo 类型的令牌时服务器端可以返回自定义令牌的令牌侦听器（令牌中存储了一组客户端与服务器端之间相互发送的令牌消息，客户端可以向服务器端发送令牌，服务器根据客户端发送的令牌类型返回对应的令牌）。

代码清单 10-2 自定义的服务器端的令牌侦听器

---

```
package serverTest;
import javax.servlet.http.HttpServlet;
import org.apache.log4j.Logger;
import org.jwebsocket.api.WebSocketPacket;
import org.jwebsocket.kit.WebSocketServerEvent;
import org.jwebsocket.listener.WebSocketServerTokenEvent;
import org.jwebsocket.listener.WebSocketServerTokenListener;
import org.jwebsocket.logging.Logging;
import org.jwebsocket.token.Token;
import org.jwebsocket.config.*;
public class JWebSocketTokenListenerSample extends HttpServlet
implements WebSocketServerTokenListener{
    private static Logger log =
        Logging.getLogger(JWebSocketTokenListenerSample.class);
    @Override
    public void processToken(WebSocketServerTokenEvent aEvent,
        Token aToken)
    {
        log.info("Client '" + aEvent.getSessionId() + "' sent Token: '" +
            aToken.toString() + "'.");
        // 可以在此处根据客户端发送的令牌类型返回相应的自定义的令牌
        String lNS = aToken.getNS();
        String lType = aToken.getType();
        // 检查客户端发送的令牌是否具有令牌类型与一个匹配的命名空间
        if (lType != null && "my.namespace".equals(lNS)) {
            // 创建一个服务器端的响应令牌
            Token lResponse = aEvent.createResponse(aToken);
            // 如果令牌类型为 getInfo，则在响应令牌中加入一些自定义消息
            if ("getInfo".equals(lType)) {
                lResponse.put("vendor", JWebSocketCommonConstants.VENDOR);
                lResponse.put("copyright",
                    JWebSocketCommonConstants.COPYRIGHT);
                lResponse.put("license", JWebSocketCommonConstants.LICENSE);
            }
            // 如果令牌类型为 my.namespace 命名空间中的其他类型，则加入一些自定义错误信息
            else

```



```

    {
        lResponse.put("code", -1);
        lResponse.put("msg", "令牌类型 '" + lType + "' 在 '" + lNS +
            "' 命名空间里不被支持.");
    }
    // 发送服务器端的响应令牌
    aEvent.sendToken(lResponse);
}

@Override
public void processClosed(WebSocketServerEvent aEvent) {
    log.info("Client '" + aEvent.getSessionId() + "' disconnected.");
}

@Override
public void processOpened(WebSocketServerEvent aEvent) {
    log.info("Client '" + aEvent.getSessionId() + "' connected.");
}

@Override
public void processPacket(WebSocketServerEvent aEvent,
    WebSocketPacket aPacket)
{
    // 可以在此处直接处理非令牌的更低阶层的数据包
}
}

```

在这段代码中，主要关注 processToken 事件处理函数中的代码。当服务器端接收到客户端发送的令牌时调用该事件处理函数。

代码的第一行自定义一个 INFO 消息，可以将该 INFO 消息直接输出到控制台，也可以将该 INFO 消息输出到日志。输出到控制台时画面显示如图 10-14 所示。

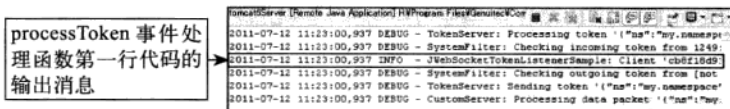


图 10-14 控制台中输出消息

代码第一行中自定义 INFO 消息的完整文字为：“Client ” + 捕获事件的 SessionId + “ sent Token: ” + 客户端发送过来的令牌消息 + “.”。

接下来获取客户端发送令牌的命名空间并赋值给变量 lNS，获取客户端发送令牌的令牌类型并赋值给变量 lType。

如果客户端发送的令牌存在令牌类型且命名空间等于“my.namespace”，则创建一个服务器端的响应令牌。

接下来判断客户端发送令牌的令牌类型。如果令牌类型等于“getInfo”，则在响应令牌中加入一些自定义消息，这些消息有如下几种。

- 1) 加入 vendor 字段，并且设置字段值为提供 jWebSocket 服务的供应商。

2) 加入 copyright 字段, 并且设置字段值为供应商所声明的版权信息。

3) 加入 license 字段, 并且设置字段值为供应商所声明的许可证信息。

当客户端接收到令牌时可以通过 aToken 参数 (表示接收到的服务器端的响应令牌) 的 vendor 属性、copyright 属性以及 license 属性访问到这些内容。

如果客户端发送的令牌的命名空间等于 “my.namespace”, 而令牌类型不等于 “getInfo”, 则在响应令牌中加入自定义错误消息, 这些消息有如下几种。

1) 加入 code 字段, 并且设置字段值为 -1。

2) 加入 msg 字段, 并且设置字段值为自定义错误信息, 文字为: “令牌类型 '” + 客户端发送令牌的令牌类型 + “' 在 '” + 客户端发送令牌的命名空间 + “' 命名空间里不被支持”。

当客户端接收到令牌时可以通过 aToken 参数的 code 属性与 msg 属性访问到这些内容。

在 processToken 事件处理函数的底部将本函数中创建的服务器端的响应令牌发送给客户端, 代码如下所示。

```
aEvent.sendToken(lResponse);
```

## 2. 在 jWebSocket 服务器中注册自定义的侦听器类

定义好侦听器类之后, 接下来的工作就是在 jWebSocket 服务器中注册所定义的侦听器类。

首先, 在 ContextListener 类中注册自定义的侦听器类 (JWebSocketTokenListenerSample), 如代码清单 10-3 所示 (代码清单 10-3 的文件名为 ContextListener.java, 包名为 serverTest)。

代码清单 10-3 在 ContextListener 类中注册自定义的侦听器类

---

```
package serverTest;
import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;
import org.jwebsocket.server.TokenServer;
import org.jwebsocket.factory.JWebSocketFactory;
public class ContextListener implements ServletContextListener{
    @Override
    public void contextInitialized(ServletContextEvent sce) {
        // 启动 jWebSocket 服务器的子系统
        JWebSocketFactory.start();
        // 获取令牌服务器
        TokenServer lServer =
            (TokenServer)JWebSocketFactory.getServer("ts0");
        // 如果获取到令牌服务器
        if( lServer != null ) {
            // 将自定义侦听器注册到服务器的侦听器链中
            lServer.addListener(new JWebSocketTokenListenerSample());
        }
    }
    @Override
```

```

    public void contextDestroyed(ServletContextEvent sce) {
        // 关闭 jWebSocket 服务器的子系统
        JWebSocketFactory.stop();
    }
}

```

---

在这段代码中，主要关注 contextInitialized 事件处理函数中的代码。

代码的第一行启动一个 jWebSocket 服务器的子系统。每一个使用 jWebSocket 服务器的 ContextListener 类都需要使用这句代码，因为在运行每一个应用程序时都需要启动一个 WebSocket 服务器的子系统。启动完毕后需要获取 jWebSocket 服务器中的令牌服务器。因为在 jWebSocket 服务器的配置文件中，org.jwebsocket.server.TokenServer（令牌服务器类）的 id 属性被配置为“ts0”，所以使用如下代码获取令牌服务器。

```

// 获取令牌服务器
TokenServer lServer = (TokenServer)JWebSocketFactory.getServer("ts0");

```

jWebSocket 服务器的配置文件中关于令牌服务器的配置代码如下所示。

```

<!-- server types to be instantiated for jWebSocket -->
<servers>
    <server>
        <name>org.jwebsocket.server.TokenServer</name>
        <id>ts0</id>
        <jar>jWebSocketTokenServer-0.10.jar</jar>
    </server>
    <server>
        <name>org.jwebsocket.server.CustomServer</name>
        <id>cs0</id>
        <jar>jWebSocketCustomServer-0.10.jar</jar>
    </server>
</servers>

```

接下来在代码中定义，如果获取到了令牌服务器，则将自定义的令牌侦听器注册到该令牌服务器的令牌侦听器链中，代码如下所示。

```

// 如果获取到令牌服务器
if( lServer != null ) {
    // 将自定义侦听器注册到服务器的侦听器链中
    lServer.addListener(new JWebSocketTokenListenerSample());
}

```

在 ContextListener 类中注册了自定义的令牌侦听器类（JWebSocketTokenListenerSample）后，还需要在 Web 工程的 web.xml 文件中添加 ContextListener 类与令牌侦听器类（JWebSocketTokenListenerSample）的定义。完成添加后的 web.xml 文件如代码清单 10-4 所示。

代码清单 10-4 添加了 ContextListener 类后的 web.xml 文件中的代码

---

```

<?xml version="1.0" encoding="UTF-8"?>

```

```

<web-app version="2.5"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
  <listener>
    <description>ServletContextListener</description>
    <listener-class>serverTest.ContextListener</listener-class>
  </listener>
  <listener>
    <description>HttpSessionListener</description>
    <listener-class>serverTest.SessionListener</listener-class>
  </listener>
  <servlet>
    <servlet-name>JWebSocketTokenListenerSample</servlet-name>
    <servlet-class>
      serverTest.JWebSocketTokenListenerSample
    </servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>JWebSocketTokenListenerSample</servlet-name>
    <url-pattern>/JWebSocketTokenListenerSample</url-pattern>
  </servlet-mapping>
  <session-config>
    <session-timeout>
      30
    </session-timeout>
  </session-config>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
</web-app>

```

现在可以直接在页面上使用自定义的令牌侦听器了。

### 3. 在页面上使用自定义的令牌侦听器

首先将代码清单 10-1 中创建的 hello\_world.html 文件复制到所创建的利用 jWebSocket 服务器进行通信的工程中。然后，在页面上添加一个测试侦听器按钮，代码如下所示。

```

<input type="button" id="btnDisConnect" onclick="btnDisConnect_click()"
value=" 关闭连接 " disabled="disabled">
<input type="button" id="btnTestListener" onclick="btnTestListener_click ()"
value=" 测试侦听器 " disabled="disabled">

```

接下来在 JavaScript 脚本代码中添加用户单击测试侦听器按钮时所调用的 btnTestListener\_click 函数，同时在有关函数中添加控制测试侦听器按钮有效性的代码。修改后的完整脚本代码如代码清单 10-5 所示。

代码清单 10-5 添加了 btnTestListener 函数的脚本代码

---

```

<script type="text/javascript" language="JavaScript">
var jWebSocketClient;
var userName;
function window_onload()
{
    if( jws.browserSupportsWebSockets() ) {
        jWebSocketClient = new jws.jWebSocketJSONClient();
        document.getElementById("btnConnect").disabled="";
    }
    else {
        var lMsg = jws.MSG_WS_NOT_SUPPORTED;
        alert( lMsg );
    }
}
function btnConnect_click()
{
    var lURL = jws.JWS_SERVER_URL;
    userName = document.getElementById("userName").value;
    var userPass = document.getElementById("userPass").value;
    var msg=document.getElementById("msg");
    msg.innerHTML=" 连接到地址: " + lURL + " 并且以 \" " + userName +
    "\" 用户名与服务器建立链接 ... " ;
    var lRes = jWebSocketClient.logon(lURL,userName,userPass, {
        OnOpen: function( aEvent ) {
            msg.innerHTML+="<br/>jWebSocket 连接已建立 " ;
        },
        OnMessage: function( aEvent, aToken ) {
            msg.innerHTML+="<br/>jWebSocket \" " +aToken.type +
            "\" 令牌收到, 消息字符串为: \" " + aEvent.data + "\" " ;
        },
        OnClose: function( aEvent ) {
            msg.innerHTML+="<br/>jWebSocket 连接被关闭. " ;
            document.getElementById("btnbroadcastText").disabled="disabled";
            document.getElementById("btnDisConnect").disabled="disabled";
            document.getElementById("btnTestListener").disabled="disabled";
        }
    });
    msg.innerHTML+="<br/>"+jWebSocketClient.resultToString(lRes);
    if(lRes.code==0)
    {
        document.getElementById("btnbroadcastText").disabled="";
        document.getElementById("btnDisConnect").disabled="";
        document.getElementById("btnTestListener").disabled="";
    }
}
function btnbroadcastText_click()
{
    var sendMsg=document.getElementById("sendMsg").value;

```

```

var msg=document.getElementById("msg");
msg.innerHTML+="  
> 广播消息: \""+sendMsg+"\""...";
var lRes = jWebSocketClient.broadcastText("",sendMsg);
if(lRes.code!=0)
    msg.innerHTML=jWebSocketClient.resultToString( lRes );
document.getElementById("sendMsg").value="";
}
function btnDisConnect_click()
{
    if(jWebSocketClient)
    {
        var msg=document.getElementById("msg");
        msg.innerHTML+="  
> 用户 \""+user+"\" 关闭连接 ";
        var lRes=jWebSocketClient.close();
        msg.innerHTML+="  
>"+jWebSocketClient.resultToString( lRes );
        if(lRes.code==0)
        {
            document.getElementById("btnbroadcastText").disabled="disabled";
            document.getElementById("btnDisConnect").disabled="disabled";
            document.getElementById("btnTestListener").disabled="disabled";
        }
    }
}
function btnTestListener_click()
{
    var lToken = {
        ns: "my.namespace",
        type: "getInfo"
    };
    jWebSocketClient.sendToken( lToken, {
        OnResponse: function( aToken ) {
            var msg=document.getElementById("msg");
            if(aToken.vendor!=null)
                msg.innerHTML+="  
> 服务器响应如下消息: " + "  
> 供应商: " +
                    aToken.vendor+ "  
> 版权: " + aToken.copyright+
                    "  
> 许可证: " + aToken.license;
            else if(aToken.code==-1)
                msg.innerHTML+="  
> 服务器响应错误消息: "+ aToken.msg;
        }
    });
}
function window_onunload()
{
    if(jWebSocketClient)
    {
        jWebSocketClient.close({timeout:3000});
    }
}
</script>

```

在这段脚本代码中，主要了解用户单击测试侦听器按钮时所调用的 `btnTestListener_click` 函数中的代码。在该函数中首先定义一个令牌，令牌的命名空间为“`my.namespace`”，令牌类型为“`getInfo`”。接下来，发送该令牌，并定义当接收到服务器端响应令牌时执行的回调函数，该函数如下所示。

```
jWebSocketClient.sendToken( lToken, {
    OnResponse: function( aToken ) {
        // 定义回调函数中执行的处理
    }
});
```

在该回调函数中，指定当接收到服务器端响应令牌时如果 `aToken` 参数（表示服务器端响应令牌）的 `vendor` 属性值不为空，则在通信消息显示区域中显示文字：“服务器响应如下消息：”+（换行标志）+“供应商：”+服务器端响应令牌的 `vendor` 属性值+（换行标志）+“版权：”+服务器端响应令牌的 `copyright` 属性值+（换行标志）+“许可证：”+服务器端响应令牌的 `license` 属性值。如果 `aToken` 参数的 `vendor` 属性值为空，则在通信消息显示区域中显示文字：“服务器响应错误消息：”+服务器端响应令牌的 `msg` 属性值（在令牌侦听器中自定义的错误消息）。

打开修改后的页面，单击建立连接按钮连接并登录到 `jWebSocket` 服务器后，再单击测试侦听器按钮，浏览器中的显示效果如图 10-15 所示。

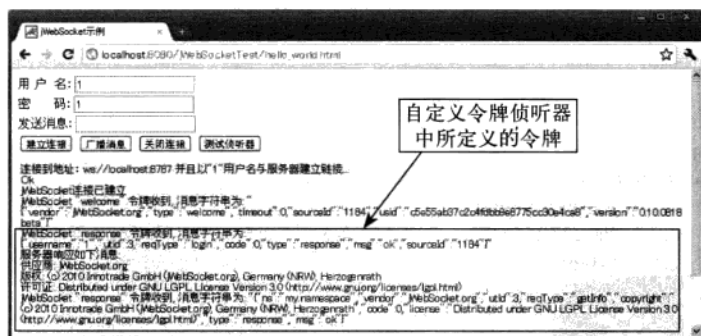


图 10-15 单击测试侦听器按钮后的显示效果

## 10.4 jWebSocket 中的令牌

前面已经介绍过，当客户端与服务器端建立连接、向服务器端发送消息以及与服务端断开连接时，服务器端都会向客户端发送一个令牌，客户端也可以直接通过 `jWebSocketJSONClient` 对象的 `sendToken` 方法向服务器端发送令牌，服务器端在接收到令牌后向客户端发送响应令牌，开发者可以通过创建令牌侦听器的方法来自定义响应令牌中的内容。本节针对这个令牌进行详细介绍。

### 10.4.1 令牌的基本概念

令牌是一个可以包含多个字段与字段值的对象。在 jWebSocketSerer-0.10.jar 包的 org.jWebSocket.token.Token 类中内置了一个 HashMap 类，该类中可以包含多个字段与字段值。在 HashMap 类中，字段名（又称键名）以字符串形式保存，字段值（又称键值）可以为任何对象。在 jWebSocket 中，令牌分为三种：JSON 格式的令牌、XML 格式的令牌与 CSV 格式的令牌，与前者不同的是，CSV 格式的令牌中只能使用简单数据类型的字段值，不能使用复合的对象结构。

接下来对这三种格式的令牌的不同作用与使用场合进行必要的介绍。

#### □ JSON 格式

JSON 格式的令牌在使用 JavaScript 语言的客户端脚本代码中可以很容易地被解析出来，而在使用 JAVA 语言的客户端中不能很容易地被解析出来。同时，使用 JSON 格式的令牌会带来一些不安全因素。因为 JSON 格式的令牌可以在客户端执行恶意代码，所以在服务器端使用 JSON 格式的令牌时要牢记将一些潜在的可执行的代码抽离出来。当浏览器客户端与具有可以用来确保安全的脚本过滤器的服务器需要大量交互时，可以使用 JSON 格式的令牌。与 JSON 格式的令牌不同的是，CSV 格式的令牌是安全的，因为它的不是由 eval 函数来解析的，而是简单地利用字符串分析器（string tokenizer）来进行解析的。

使用示例：

```
{field1:value1;field2:"value2";field3:[arrayitem1,arrayitem2];field4:{objfield1:valuel;objfield2:value2}}
```

#### □ CSV 格式

CSV 是一种简便易用的格式。但是，它的一个缺点是不支持复合的对象结构，只支持使用简单数据类型的行结构。要交换大量的简单类型的数据，CSV 是最好的选择。

使用示例：

```
field1=value1,field2="value2",field3=value3
```

#### □ XML 格式

XML 格式是一种最灵活也是最冗长的数据格式。XML 格式中可以包含任意的对象类型。因此在不需要交换大量的数据而又想灵活使用令牌中包含的数据时，XML 格式是最好的选择。

目前 XML 格式的令牌在 1.0 及之前版本的 jWebSocket 中没有使用，预计在 1.1 版中将会使用。

下面对 jWebSocket 中使用到的与令牌相关的几个术语进行介绍。

#### □ 令牌类型

每个令牌都具有一个令牌类型，该令牌类型被保存在其 type 字段中。令牌类型决定了令牌可以拥有哪些其他字段，稍后将对此进行详细解释。



### □ 命名空间

在 jWebSocket 中，各种强大的通信能力是依靠各种服务器端或客户端的插件来实现的。可以通过自定义插件来扩展 jWebSocket 中现有的通信功能。为了避免各种令牌中字段名冲突，jWebSocket 提供了一种使用命名空间来管理令牌的机制。可以将各种自定义的令牌指定在不同的命名空间中，就像 10.3.2 节中将客户端自定义的令牌指定在“my.namespace”中那样。一个插件在启动并读取令牌的内容时，首先会检查令牌的命名空间与插件的命名空间是否匹配。

### □ 令牌 id

每一个令牌都在 session 中拥有一个唯一的 id，客户端与服务器端在交换令牌（即客户端发送令牌，服务器端响应令牌）的时候会交换令牌 id。基于 jWebSocket 服务器的多线程处理机制的特性，jWebSocket 框架并不保证客户端提出请求的顺序与它随后接收到的服务器端的响应的顺序是一致的。因此客户端向服务器端发送请求时，会为每一个新的令牌分配一个新 session 中唯一的令牌 id。服务器端在作出响应时会接管这个令牌 id 并将其放在响应令牌中，这样客户端就能将每一个接收到的响应分配给它之前发送的请求。

但是，不需要在开发应用程序时关注令牌 id。所有客户端接收到的响应都可以在 OnMessage 回调函数中被捕获，因此多数时候可以不必关注令牌 id。

## 10.4.2 系统令牌

接下来看一下 jWebSocket 的 SystemPlugIn 中支持的令牌，首先介绍服务器端发送给客户端的令牌。

### 1. 服务器端发送给客户端的令牌

#### □ welcome 令牌

当客户端与服务器端之间建立连接后，服务器端将 welcome 令牌发送给客户端。welcome 令牌是唯一一个将标识每个客户端的唯一 session id 发送给客户端的令牌。在整个会话期间，服务器端不会再次将该 session id 发送给客户端，客户端也不会再次向服务器端请求其 session id。在任何情况下，只要客户端与服务器端之间建立连接，总会执行该客户端的 processOpened 方法。

welcome 令牌中所包含的字段及其说明如表 10-2 所示。

表 10-2 welcome 令牌中所包含的字段及其说明

字段	数据类型	说明
type	String	始终为“welcome”
vendor	String	提供 jWebSocket 服务的供应商的名字。可以在 Config.java 文件中对其进行设置
version	String	jWebSocket 服务器的版本号
usid	String	jWebSocket 服务器端用来标识每个客户端的唯一 session id
sourceId	Integer	每个客户端的唯一 id，可以用来标识不同的客户端
timeout	Integer	以毫秒为单位的会话超时时间。如果在此期间客户端一直处于非活动状态，服务器端将自动关闭服务器端与该客户端之间的连接

### □ goodBye 令牌

当客户端向服务器端发出关闭连接的请求后，服务器端将 goodBye 令牌作为响应令牌发送给客户端。采用 JavaScript 脚本语言的客户端的 close 方法支持一个 timeout 选项。如果 timeout 的值  $\leq 0$ ，客户端与服务器端之间的连接将被立即关闭。如果 timeout 的值  $> 0$ ，客户端将向服务器端发送一个 close 令牌并为 goodBye 响应令牌等待 timeout 值中指定的时间，在这种情况下，服务器端将在发出 goodBye 响应令牌后关闭连接。如果客户端在 timeout 值所指定的时间范围内没有收到 goodBye 响应令牌，客户端也将关闭与服务器端的连接。在任何情况下，只要客户端与服务器端的连接被中止，都会执行该客户端的 processClosed 方法。

goodBye 令牌中所包含的字段及其说明如表 10-3 所示。

表 10-3 goodBye 令牌中所包含的字段及其说明

字段	数据类型	说明
type	String	始终为“goodBye”
vendor	String	提供 jWebSocket 服务的供应商的名字。可以在 Config.java 文件中对其进行设置
version	String	jWebSocket 服务器的版本号
usid	String	jWebSocket 服务器端用来标识每个客户端的唯一 session id
port	Integer	服务器端与客户端之间建立连接时所使用的 TCP 端口号

### □ response 令牌

当客户端向服务器端或其他客户端发出请求后，服务器端或其他客户端将 response 响应令牌发送给该客户端。

response 令牌中所包含的字段及其说明如表 10-4 所示。

表 10-4 response 令牌中所包含的字段及其说明

字段	数据类型	说明
type	String	始终为“response”
utid	Integer	作为响应而发送给发出请求的客户端的唯一的令牌 id
reqType	String	作为响应而发送给发出请求的客户端的令牌类型，总是等于客户端请求令牌的类型
code	Integer	当客户端请求被正确执行时返回结果为 0。执行发生错误时为对应的错误号
result	Variant	客户端请求的执行结果，可以为任意类型（依客户端的请求及其所调用的服务器端或其他客户端的方法而定）
msg	String	客户端请求在执行过程中发生错误时的错误信息描述

### □ event 令牌

event 令牌是服务器或其他客户端作为消息主动发送给客户端的令牌。当其他客户端建立或关闭与 jWebSocket 服务器之间的连接，或者当服务器端的会话超时，将主动关闭与客户端之间的连接时会触发事件，向客户端发送 event 令牌。

event 令牌中所包含的字段及其说明如表 10-5 所示。

表 10-5 event 令牌中所包含的字段及其说明

字段	数据类型	说明
type	String	始终为 “event”
name	String	事件名，该令牌中的其他字段依不同的事件而定
...	...	其他字段，依不同的事件而定

❑ connect event 令牌

当客户端接收到 connect event 令牌时，表示有一个新的客户端与 jWebSocket 服务器之间建立了连接。这个事件是可选的，也是可配置的。如果配置了该事件，则一个客户端与 jWebSocket 服务器之间建立连接时，jWebSocket 服务器将向当前所有与自己处于连接状态的其他客户端发送 connect event 令牌。

connect event 令牌中所包含的字段及其说明如表 10-6 所示。

表 10-6 connect event 令牌中所包含的字段及其说明

字段	数据类型	说明
type	String	始终为 “event”
name	String	始终为 “connect”
sourceId	String	与服务器端建立连接的客户端的唯一 id( 可选字段，可以在服务器端配置是否广播 sourceId 字段值 )
clientCount	Integer	当前与服务器端处于连接状态的客户端数 ( 可选字段，可以在服务器端配置是否广播 clientCount 字段值 )

❑ disconnect event 令牌

当客户端收到 disconnect event 令牌时，表示有一个客户端与 jWebSocket 服务器之间的连接被关闭。这个事件是可选的，同时也是可配置的。如果配置了该事件，则一个客户端与 jWebSocket 服务器之间建立的连接被关闭时，jWebSocket 服务器将向当前所有与自己处于连接状态的其他客户端发送 disconnect event 令牌。

disconnect event 令牌中所包含的字段及其说明如表 10-7 所示。

表 10-7 disconnect event 令牌中所包含的字段及其说明

字段	数据类型	说明
type	String	始终为 “event”
name	String	始终为 “disconnect”
sourceId	String	与服务器的连接被关闭的客户端的唯一 id( 可选字段，可以在服务器端配置是否广播 sourceId 字段值 )
clientCount	Integer	当前与服务器端处于连接状态的客户端数 ( 可选字段，可以在服务器端配置是否广播 clientCount 字段值 )

❑ login event 令牌

当客户端收到 login event 令牌时，表示有一个新的客户端登录到 jWebSocket 服务器。

这个事件可以用于实时更新客户端的用户列表。这个事件是可选的，也是可配置的。如果配置了该事件，则一个客户端登录到 jWebSocket 服务器时，jWebSocket 服务器将向其他所有当前与自己处于连接状态的客户端发送 login event 令牌。

login event 令牌中所包含的字段及其说明如表 10-8 所示。

表 10-8 login event 令牌中所包含的字段及其说明

字段	数据类型	说明
type	String	始终为“event”
name	String	始终为“login”
sourceId	String	登录到服务器的客户端的唯一 id(可选字段，可以在服务器端配置是否广播 sourceId 字段值)
clientCount	Integer	当前与服务器端处于连接状态的客户端数(可选字段，可以在服务器端配置是否广播 clientCount 字段值)
userName	String	登录到服务器的客户端所使用的用户名

#### □ logout event 令牌

当客户端接收到 logout event 令牌时，表示有一个新的客户端退出了 jWebSocket 服务器。这个事件可以用来实时更新客户端的用户列表。这个事件是可选的，也是可配置的。如果配置了该事件，则一个客户端退出 jWebSocket 服务器时，jWebSocket 服务器将向当前所有与自己处于连接状态的其他客户端发送 logout event 令牌。

logout event 令牌中所包含的字段及其说明如表 10-9 所示。

表 10-9 logout event 令牌中所包含的字段及其说明

字段	数据类型	说明
type	String	始终为“event”
name	String	始终为“logout”
sourceId	String	退出服务器的客户端的唯一 id(可选字段，可以在服务器端配置是否广播 sourceId 字段值)
clientCount	Integer	当前与服务器端处于连接状态的客户端数(包括退出服务器的客户端在内，可选字段，可以在服务器端配置是否广播 clientCount 字段值)
userName	String	退出服务器的客户端所使用的用户名

## 2. 客户端发送给服务器端的令牌

接下来看一下客户端发送给服务器端的令牌。在大多数情况下，客户端发送了令牌之后(可以将发送令牌的动作视为发出一个命令，通知服务器执行某些处理)，服务器端会发送一个 response 响应令牌(除非由于某些特殊的原因而显式指定服务器端不需要发送 response 响应令牌)，客户端请求的执行结果将被包含在 response 响应令牌的结果字段值、code 字段值与 msg 字段值中。因为 JavaScript 脚本不支持异步调用，所以它会为每个令牌提供一个可选的 OnResponse 侦听器。

### □ login 令牌

客户端与服务端建立连接后登录到服务器，通知服务器对其进行验证。客户端将处于等待状态，直到服务器端发送响应令牌，通知客户端登录状态或潜在错误。一个用户已经通过验证并登录到服务器后，如果该客户端再次发送 login 令牌，之前已经登录的用户将会自动退出。

login 令牌中所包含的字段及其说明如表 10-10 所示。

表 10-10 login 令牌中所包含的字段及其说明

字段	数据类型	说明
type	String	始终为“login”
ns	String	命名空间为“org.jWebSocket.plugins.system”
username	String	之前被自动退出服务器的用户名
password	String	字段值根据服务器端设置的安全级别来决定（可能是自动退出服务器的用户的密码）

### □ logout 令牌

当前用户退出 jWebSocket 服务器，并没有关闭客户端与服务端端的连接。可以指定是否允许其他用户利用同一个连接登录到 jWebSocket 服务器。

logout 令牌中所包含的字段及其说明如表 10-11 所示。

表 10-11 logout 令牌中所包含的字段及其说明

字段	数据类型	说明
type	String	始终为“logout”
ns	String	命名空间为“org.jWebSocket.plugins.system”

### □ close 令牌

允许采用 JavaScript 脚本语言的客户端的 close 方法使用一个 timeout 可选参数。如果 timeout 参数值  $\leq 0$ ，客户端与服务端之间的连接将立即被关闭。如果 timeout 参数值  $> 0$ ，客户端将向服务器端发送一个 close 令牌，并且处于等待状态，直到接收到服务器端的 goodBye 响应令牌。在这种情况下，服务器端在发送了 goodBye 响应令牌之后关闭与客户端之间的连接。如果客户端在 timeout 参数值指定的时间内没有接收到 goodBye 响应令牌，客户端也将关闭与服务端端的连接。

close 令牌中所包含的字段及其说明如表 10-12 所示。

表 10-12 close 令牌中所包含的字段及其说明

字段	数据类型	说明
type	String	始终为“close”
ns	String	命名空间为“org.jWebSocket.plugins.system”
timeout	Integer	以毫秒为单位的超时时间

### □ send 令牌

客户端在 send 令牌的 targetId 字段值中指定其他客户端 id 并将其发送给服务端，服务器端会将接收到的 send 令牌转发给在 targetId 字段值中指定的其他客户端。可以在 send 令牌的 responseRequested 字段中指定是否要求该客户端（接收 send 令牌的客户端）作出响应（确认已接收到服务器端转发的 send 令牌）。如果服务器端没有找到 targetId 字段值中指定的客户端，服务器端将向发送 send 令牌的客户端发送一个 response 令牌，并且在其中包含错误信息。

在指定其他客户端时，不能利用该客户端登录 jWebSocket 服务器端时使用的用户名来进行指定，只能利用该客户端的标识 id 来进行指定。因为只要 Web 应用程序允许，可以在多个浏览器中，同一个浏览器的多个标签中，不同的客户端计算机（或移动设备）中使用相同的用户名进行登录（除非开发者在 Web 应用程序中进行了特别处理，禁止使用相同的用户名进行登录）。如果一个客户端只需要接收服务器端发送的消息，那么它不需要登录到 jWebSocket 服务器中（只需与服务器建立连接即可）。

send 令牌中所包含的字段及其说明如表 10-13 所示。

表 10-13 send 令牌中所包含的字段及其说明

字段	数据类型	说明
type	String	始终为“send”
ns	String	命名空间为“org.jWebSocket.plugins.system”
data	String	发送的消息字符串
sourceId	String	发送消息的客户端的 id
targetId	String	接受消息的客户端的 id
responseRequested	Boolean	指定接受消息的客户端是否要做出响应
sender	String	发送消息的客户端的用户名

### □ broadcast 令牌

客户端向服务器端发送 broadcast 令牌后，服务器端将该令牌广播给所有与之相连接的客户端。可以指定是否因为某种特殊原因而将令牌广播给发送 broadcast 令牌的客户端自身。可以指定服务器端是否向广播消息的客户端发送响应令牌，响应令牌的内容取决于其他客户端对广播令牌做出的响应结果。广播令牌可以在聊天室网站中用来广播聊天内容，可以在游戏网站中用来向所有玩家分配游戏角色等，也可以在服务器被配置成不自动发送 connect、disconnect、login 与 logout 事件令牌时用来将某个客户端与服务器端的交互操作通知给其他所有的客户端。

broadcast 令牌中所包含的字段及其说明如表 10-14 所示。

表 10-14 broadcast 令牌中所包含的字段及其说明

字段	数据类型	说明
type	String	始终为“broadcast”
ns	String	命名空间为“org.jWebSocket.plugins.system”
data	String	发送的消息字符串
sourceId	String	发送消息的客户端的 id
senderIncluded	Boolean	是否将消息发送给发送消息的客户端自身（默认为 false）
responseRequested	Boolean	指定接受消息的客户端是否希望接收响应令牌
sender	String	发送消息的客户端的用户名

### □ echo 令牌

客户端可以通过发送 echo 令牌来向服务器端发送一个消息。客户端期望得到的服务器端发送的响应令牌中包含相同的数据。通常 Web 应用程序中不会使用这个令牌，除非 Web 应用程序需要进行连接测试或性能测试。

echo 令牌中所包含的字段及其说明如表 10-15 所示。

表 10-15 echo 令牌中所包含的字段及其说明

字段	数据类型	说明
type	String	始终为“echo”
ns	String	命名空间为“org.jWebSocket.plugins.system”
data	String	需要发送给服务器端并返回的数据

### □ ping 令牌

ping 令牌中包含了一个客户端发送给服务器端的简单而短小的消息，该消息仅仅用来声明客户端依然处于活动状态。如果服务器端在 timeout 指定的会话超时时间内没有接收到客户端的任何数据，超过 timeout 指定的超时时间后服务器端将主动关闭与客户端之间的连接。

ping 令牌中所包含的字段及其说明如表 10-16 所示。

表 10-16 ping 令牌中所包含的字段及其说明

字段	数据类型	说明
type	String	始终为“ping”
ns	String	命名空间为“org.jWebSocket.plugins.system”
echo	Boolean	指定服务器端是否需要做出响应（默认为 false）。只有客户端需要检查服务器端是否处于活动状态时才有必要将 echo 字段值设定为 true

### □ getClients 令牌

客户端使用 getClients 令牌来向服务器端请求与之连接的客户端名单。可以使用一个 mode 可选项（默认为 0）来指定服务器端是否返回所有与之相连接的客户端名单、只返回所有已登录到服务器的客户端名单与只返回所有只连接而没有登录到服务器的客户端名单。

response 响应令牌的 result 字段值为一个数组，该数组中的每个数组项的内容为一串字符串，文字为：已登录到服务器的客户端所使用的用户名或虚线（表示该客户端只与服务器端建立了连接，并没有登录到服务器）+ “@” + 客户端的 id。

getClient 令牌中所包含的字段及其说明如表 10-17 所示。

表 10-17 getClient 令牌中所包含的字段及其说明

字段	数据类型	说明
type	String	始终为 “getClient”
ns	String	命名空间为 “org.jWebSocket.plugins.system”
mode	Integer	0: 请求服务器端返回所有的与之相连接的客户端名单 1: 请求服务器端返回所有已登录到服务器的客户端名单 2: 返回所有只连接而没有登录到服务器的客户端名单

## 10.5 jWebSocket 中服务器端的插件

本节介绍 jWebSocket 中服务器端的插件的基础知识及创建方法。首先对 jWebSocket 中服务器端的插件进行详细介绍。

### 10.5.1 服务器端插件的基础知识

#### 1. 什么是服务器端的插件

在 jWebSocket 中，几乎所有的高端处理都是依靠插件来实现的，因此 jWebSocket 服务器提供了所有必需的管理能力，其中之一就是插件链的实现机制。在这个插件链中，多个插件按照一定的顺序进行排列，这个顺序是可以任意改变，新的插件可以很容易地被插入进去。一般来说，一个插件可以处理任意数量的从客户端发出的令牌中包含的各种动态字段值中解析出来的“命令”。一个复合插件可以由多个不同的可以在运行时被置入或动态链接入的包或者 jar 文件中的多个类所组成。

插件与插件链可以使用户很容易地根据自己的需要在 jWebSocket 已经提供的功能的基础上实现新的 WebSocket 功能。插件链的示意图如图 10-16 所示。

#### 2. 令牌在服务器中的处理流程

当调用 jWebSocket 中的一个 server（服务器）类的构造器来创建对象时，也将创建一个 PlugInChain（插件链）类的对象。例如，在创建一个 TokenServer（令牌服务器）对象的时候，也将创建一个 TokenPlugInChain 类（一个实现了 PlugInChain 接口的 BasePlugInChain 抽象类的子类）的对象。

当客户端发送一个数据包或令牌时，该数据包或令牌首先被服务器端的一个连接器所接收，该连接器通过引擎将数据包或令牌转发给服务器，如图 10-16 所示。服务器将令牌转发给插件链后，令牌在插件链的所有插件中进行遍历，直到被处理它的目标插件接收到为止。



通常目标插件处理完毕后令牌的遍历过程也就结束了，插件决定是否向客户端做出响应或回答。

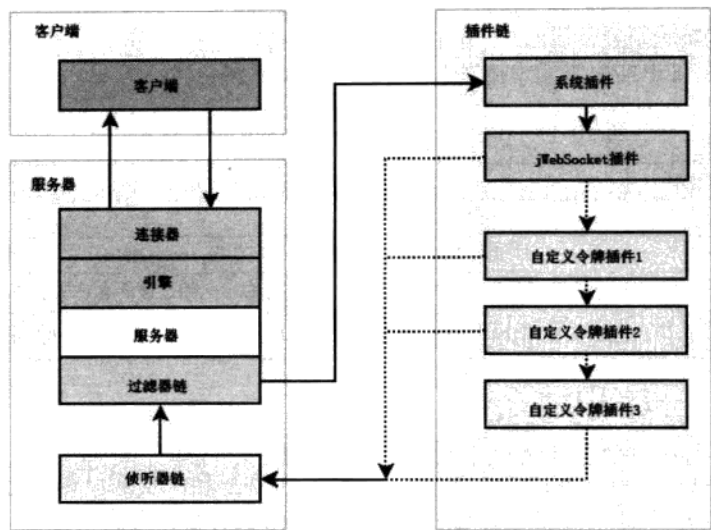


图 10-16 插件链示意图

3. 处理令牌

每一个插件都属于一个唯一的命名空间。插件通常会检查接收到的令牌的命名空间与自身所属的命名空间是否一致。如果一致则对令牌的类型（type 字段值）进行检查。令牌的类型可以被理解为客户端向服务器端发出的一个命令。

可以对 jWebSocket 中已提供的插件进行扩展，或者添加新的插件。

通常对一个 TokenPlugIn 类进行扩展时需要实现的方法如表 10-18 所示（不包括实现 TokenPlugIn 插件本身功能的专有方法在内）。

表 10-18 对一个 TokenPlugIn 类进行扩展时需要实现的方法

方法	说明
processToken	当一个客户端向服务器端发送一个消息时，TokenPlugInChain 类总会调用 TokenPlugIn 类的 processToken 方法，消息数据将作为对象传递给该方法。消息数据可以被转换成底层协议所支持的任何数据类型
processPacket	当一个客户端向服务器发送一个消息时，BasePlugInChain 类总会调用 TokenPlugIn 类的 processPacket 方法，消息数据将作为对象传递给该方法。消息数据可以被转换成底层协议所支持的任何数据类型
engineStarted	启动引擎时服务器总会调用 TokenPlugIn 类的 engineStarted 方法。可以在这个方法中初始化插件。在极个别情况下，需要使用多个引擎，一定注意不要在无意中对一个插件进行两次初始化工作
engineStopped	停止引擎时服务器总会调用 TokenPlugIn 类的 engineStopped 方法。可以在这个方法中执行插件内部的资源回收工作。在极个别情况下，需要使用多个引擎，一定注意不要在无意中对一个插件进行两次资源回收工作

(续)

方法	说明
connectorStarted	当一个新的客户端与 jWebSocket 服务器建立连接时, 插件链总会调用 TokenPlugIn 类的 connectorStarted 方法。如果一个客户端需要自动注册在某个特定的数据流中时, 可以在 connectorStarted 方法中指定这个处理
connectorStopped	当一个客户端与 jWebSocket 服务器之间的连接被关闭时, 插件链总会调用 TokenPlugIn 类的 connectorStopped 方法。如果一个客户端需要自动从某个特定的数据流中取消注册时, 可以在 connectorStopped 方法中指定这个处理

## 10.5.2 创建自定义服务器端插件

接下来介绍如何创建一个服务器端的 jWebSocket 插件。在 jWebSocket 中, 插件通过处理客户端发送的消息来扩展 jWebSocket 服务器端的功能。客户端发送的消息经过 jWebSocket 过滤器链过滤之后确保了与侦听器相同级别的安全性。

可以编写代码来加载插件 (尤其是出于开发、测试等目的), 也可以在 jWebSocket.xml 文件中引用插件以便在运行时将它动态加载 (推荐在实际应用的 Web 系统中或者在发布插件时采用这种方法)。

插件通常是被用来创建而不是用来实现某种应用程序中的特定逻辑。如果要实现某种专有逻辑, 可以使用 jWebSocket 中的侦听器。当然, 也可以根据需要在插件中实现任何逻辑。与内置在应用程序中的侦听器不同的是, 使用插件的最大好处是可以把插件视为软件中独立的一个部分, 可以单独发布插件, 也可以在多个应用程序中共享一个插件。

在 Web 网站或应用程序中创建自定义插件并且在其中实现自定义功能是一个简单的过程, 分为如下几步:

- 1) 创建一个服务器端的插件。
- 2) 在 jWebSocket 服务器端添加这个插件。
- 3) 创建一个客户端插件 (推荐使用这种方法来保证模块的独立性)。
- 4) 在 Web 页面中使用插件。

### 1. 创建一个服务器端插件

对 jWebSocket 服务器的功能进行扩展的第一步就是创建一个服务器端的插件。一个插件通常需要继承 jWebSocket 中的 TokenPlugIn 类。推荐将开发的自定义插件放在一个独立的包中, 这样以后可以将它放在一个独立的 .jar 文件中, 并且将该文件放在每一个 jWebSocket 的服务器中。

代码清单 10-6 为一个创建服务器端插件的代码示例, 该代码示例的文件名为 SamplePlugIn.java 文件, 使用的包名为 plugInTest。

代码清单 10-6 创建服务器端插件的代码示例

```
package plugInTest;
import java.util.Date;
```

```

import java.util.List;
import java.util.Map;
import javolution.util.FastList;
import javolution.util.FastMap;
import org.apache.log4j.Logger;
import org.jwebsocket.api.WebSocketConnector;
import org.jwebsocket.api.WebSocketEngine;
import org.jwebsocket.config.JWebSocketServerConstants;
import org.jwebsocket.kit.CloseReason;
import org.jwebsocket.kit.PlugInResponse;
import org.jwebsocket.logging.Logging;
import org.jwebsocket.plugins.TokenPlugIn;
import org.jwebsocket.token.Token;
public class SamplePlugIn extends TokenPlugIn{
    private static Logger log=Logging.getLogger(SamplePlugIn.class);
    private static String NS_SAMPLE = JWebSocketServerConstants.NS_BASE +
        ".plugins.samples";
    private static String SAMPLE_VAR=NS_SAMPLE+".started";
    public SamplePlugIn()
    {
        if(log.isDebugEnabled())
            log.debug("Instantiating sample plug-in...");
        // 指定示例插件的命名空间
        this.setNamespace(NS_SAMPLE);
    }
    @Override
    public void connectorStarted(WebSocketConnector aConnector)
    {
        // 当客户端与服务器端建立连接时将调用这个方法
        aConnector.setVar(SAMPLE_VAR, new Date().toString());
    }
    @Override
    public void connectorStopped(WebSocketConnector aConnector,
        CloseReason aCloseReason) { // 当客户端与服务器端的连接被关闭时将调用这个方法
    }
    @Override
    public void engineStarted(WebSocketEngine aEngine){
        super.engineStarted(aEngine);
    }
    @Override
    public void engineStopped(WebSocketEngine aEngine){
        super.engineStopped(aEngine);
    }
    @Override
    public void processToken(PlugInResponse aResponse,
        WebSocketConnector aConnector,Token aToken){
        // 获取令牌类型
        // 类型可以被理解为一个“命令”
        String lType=aToken.getType();
        // 获取令牌命名空间
        // 每个插件都有自己的唯一命名空间

```

```

String lNS=aToken.getNS();
// 检查令牌是否拥有类型和一个与插件相匹配的命名空间
if(lType!=null&&lNS!=null&&lNS.equals(getNamespace())){
    // 获取服务器上的系统时间
    if(lType.equals("requestServerTime")){
        // 创建响应令牌
        // 响应令牌中包括唯一的令牌 id
        Token lResponse=createResponse(aToken);
        // 添加 time 字段与 started 字段
        lResponse.put("time",new Date().toString());
        lResponse.put("started",aConnector.getVar(SAMPLE_VAR));
        // 向客户端发送响应令牌
        sendToken(aConnector,aConnector,lResponse);
    }
}
}
}

```

在这个代码示例中，主要关注 processToken 事件函数中的代码。当服务器接收到客户端发送的令牌时，TokenPlugInChain 类将会调用该事件函数来处理令牌中的数据。在 processToken 事件函数的开头获取令牌的类型与令牌的命名空间，令牌的类型可以被理解成客户端向服务器端发送的，要求实现某种功能的一个“命令”。如果令牌没有类型或者命名空间与插件的命名空间不匹配，则退出事件函数。如果令牌拥有类型且令牌的命名空间与插件的命名空间匹配，则检查令牌类型是否为“requestServerTime”，如果不是则退出事件函数，如果是则创建一个响应令牌，获取服务器上的系统时间并将其放入令牌的 time 字段中，同时将客户端与服务器端建立连接的时间放入令牌的 started 字段中，然后将该响应令牌发送回客户端。

## 2. 在 jWebSocket 服务器端的插件链中添加自定义插件

创建了自定义插件之后需要将这个插件放入 jWebSocket 服务器端的插件链中，方法如下。

```

// 将自定义插件放入 jWebSocket 服务器端的插件链中
TokenServer lServer = (TokenServer)JWebSocketFactory.getServer("ts0");
SamplePlugIn lsp = new SamplePlugIn();
lServer.getPlugInChain().addPlugIn(lsp);

```

代码清单 10-7 为一个放入了自定义插件的 ContextListener 类的完整代码，代码文件为 ContextListener.java 文件，包名为 serverTest。

代码清单 10-7 放入了自定义插件的 ContextListener 类的完整代码

```

package serverTest;
import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;
import org.jwebsocket.server.TokenServer;
import org.jwebsocket.factory.JWebSocketFactory;

```

```

import pluginTest.SamplePlugin;
public class ContextListener implements ServletContextListener{
    @Override
    public void contextInitialized(ServletContextEvent sce) {
        // 启动 jWebSocket 服务器的子系统
        JWebSocketFactory.start();
        // 获取令牌服务器
        TokenServer lServer = (TokenServer)JWebSocketFactory.getServer("ts0");
        // 如果获取到令牌服务器
        if( lServer != null ) {
            SamplePlugin ISP=new SamplePlugin();
            lServer.getPluginChain().addPlugin(ISP);
        }
    }
    @Override
    public void contextDestroyed(ServletContextEvent sce) {
        // 关闭 jWebSocket 服务器的子系统
        JWebSocketFactory.stop();
    }
}

```

在 contextInitialized 事件函数中，通过 JWebSocketFactory 类来装载并启动 jWebSocket 的服务器，同时加载所有该服务器需要使用的类库、默认插件与过滤器。加载完毕后通过令牌服务器的 id 来获取令牌服务器。在 jWebSocket.xml 配置文件中配置令牌服务器的 id 为“ts0”，配置代码如下所示。

```

<servers>
  <server>
    <name>org.jwebsocket.server.TokenServer</name>
    <id>ts0</id>
    <jar>jWebSocketTokenServer-0.10.jar</jar>
  </server>
  <server>
    <name>org.jwebsocket.server.CustomServer</name>
    <id>cs0</id>
    <jar>jWebSocketCustomServer-0.10.jar</jar>
  </server>
</servers>

```

接下来创建自定义插件的一个实例并且将其添加到令牌服务器的插件链中。现在，客户端可以使用这个自定义插件了。

### 3. 创建客户端插件

一般可以通过两种途径来访问服务器端的插件。一是使用 jWebSocket.js 这个 JavaScript 类库文件中的 sendToken 方法，并且为它的 OnMessage 方法实现一个侦听器。这种方法与 10.3 节中介绍的在页面上使用 sendToken 方法发送令牌的处理十分相似，故不再赘述。

建议采用第二种途径，即为客户端插件定义一个独立的 JavaScript 脚本文件，这样可以

确保插件的独立性，使将来发布插件时所要的工作变得更加容易（只需发布服务器端的插件代码文件与客户端的插件代码文件）。

代码清单 10-8 为一个客户端插件的脚本文件示例。它为 jWebSocketTokenClient 对象提供了一个 processToken 函数、一个 requestServerTime 函数与一个 setSamplesCallbacks 函数。

代码清单 10-8 客户端插件的脚本文件示例

---

```

jws.SamplesPlugIn = {
    // 设置插件的命名空间
    // 如果需要修改这个命名空间，要同时修改服务器端插件的命名空间
    NS: jws.NS_BASE + ".plugins.samples",
    processToken: function( aToken ) { // 检查命名空间是否匹配
        if( aToken.ns == jws.SamplesPlugIn.NS ) {
            // 如果有必要，可以直接处理接收到的服务器端的响应令牌中的数据
            if( "requestServerTime" == aToken.reqType ) {
                // 此处功能仅作演示用
                if( this.OnSamplesServerTime ) {
                    this.OnSamplesServerTime( aToken );
                }
            }
        }
    },
    requestServerTime: function( aOptions ) {
        var lRes = this.createDefaultResult();
        if( this.isConnected() ) {
            var lToken = {
                ns: jws.SamplesPlugIn.NS,
                type: "requestServerTime"
            };
            this.sendToken( lToken, aOptions );
        }
        else
        {
            lRes.code = -1;
            lRes.localeKey = "jws.jsc.res.notConnected";
            lRes.msg = "客户端与服务器端之间的连接未建立。";
        }
        return lRes;
    },
    setSamplesCallbacks: function( aListeners ) {
        if( !aListeners ) {
            aListeners = {};
        }
        if( aListeners.OnSamplesServerTime !== undefined ) {
            this.OnSamplesServerTime = aListeners.OnSamplesServerTime;
        }
    }
}

// 在 TokenClient 类中添加共享的插件对象
jws.oop.addPlugIn( jws.jWebSocketTokenClient, jws.SamplesPlugIn );

```

---

在定义客户端插件的脚本代码的开头，需要定义客户端插件的命名空间。注意，该命名空间必须与服务器端的插件的命名空间保持一致。

当客户端接收到服务器端的令牌时，调用 `processToken` 函数。在该函数中首先检查服务器端令牌的命名空间与客户端插件的命名空间是否一致，如果不一致，则直接退出函数，不执行任何处理；如果一致，则检查令牌的 `reqType` 字段（该字段与客户端发出令牌的 `type` 字段始终保持一致，代表客户端发出令牌的类型，可以理解为客户端向服务器端发出的“命令”）是否为“`requestServerTime`”，如果是则判断插件中有没有设定 `OnSamplesServerTime` 事件的回调函数，如果已经指定则执行页面上指定的回调函数。

`requestServerTime` 函数为一个可以被页面上脚本代码直接调用的函数。当用户单击页面上的某个按钮（例如单击测试插件按钮）时将会调用这个函数。这个函数首先调用 `jWebSocketJSONClient` 对象的 `createDefaultResult` 方法创建一个执行结果对象，然后判断客户端与服务器端是否已建立连接，如果已建立连接，则创建一个令牌，令牌的命名空间为本插件的命名空间，令牌的类型（通过 `type` 字段值来设定）为“`requestServerTime`”，之后向服务器端发送这个令牌；如果客户端与服务器端未建立连接，则将执行结果对象的 `code` 属性值设为 `-1`（表示执行操作失败），`localeKey` 属性值设为“`jws.jsc.res.notConnected`”，`msg` 属性值设为“客户端与服务器端之间的连接未建立。”，然后将执行结果对象返回给前台页面的 JavaScript 脚本代码。

在 `setSamplesCallbacks` 函数中，设定 `OnSamplesServerTime` 事件的回调函数，当侦听器侦听到服务器端发出的响应令牌时，调用页面上指定的回调函数。

#### 4. 在页面上调用客户端插件

创建好了客户端插件之后，使用服务器端插件的最后一步是在页面上调用客户端插件。调用的第一步为在页面上引用客户端插件的脚本代码文件，代码如下所示（注意不要颠倒插件脚本文件与 `jWebSocket` 脚本库文件的先后顺序）。

```
<script type="text/javascript" src="<url>/res/js/jWebSocket.js"></script>
<script type="text/javascript" src="<url>/res/js/jwsSamplesPlugIn.js">
</script>
```

接下来在代码清单 10-1 所创建的页面的基础上，添加一个测试插件按钮，单击该按钮后将调用客户端插件来获取服务器的系统时间并将其显示在页面上。添加测试插件按钮的 HTML 代码如下所示。

```
<input type="button" id="btnDisConnect" onclick="btnDisConnect_click()"
value=" 关闭连接 " disabled="disabled">
<input type="button" id="btnTestPlugIn" onclick="btnTestPlugIn_click()"
value=" 测试插件 " disabled="disabled">
```

接下来在 JavaScript 脚本代码中添加用户单击测试插件按钮时所调用的 `btnTestPlugIn_click` 函数。在有关函数中添加控制测试插件按钮有效性的代码，以及其他使用插件时必须要用到的代码，修改后的完整脚本代码如代码清单 10-9 所示。

代码清单 10-9 在页面上使用客户端插件的完整脚本代码

```

<script type="text/javascript" language="JavaScript">
var jWebSocketClient;
var userName;
function window_onload()
{
    if( jws.browserSupportsWebSockets() ) {
        jWebSocketClient = new jws.jWebSocketJSONClient();
        jWebSocketClient.setSamplesCallbacks(
            {OnSamplesServerTime:getServerTimeCallback}
        );
        document.getElementById("btnConnect").disabled="";
    }
    else {
        var lMsg = jws.MSG_WS_NOT_SUPPORTED;
        alert( lMsg );
    }
}
function btnConnect_click()
{
    var lURL = jws.JWS_SERVER_URL;
    userName = document.getElementById("userName").value;
    var userPass = document.getElementById("userPass").value;
    var msg=document.getElementById("msg");
    msg.innerHTML=" 连接到地址: " + lURL + " 并且以 \" " + userName +
    "\" 用户名与服务器建立链接 ... " ;
var lRes = jWebSocketClient.logon(lURL,userName,userPass, {
    OnOpen: function( aEvent ) {
        msg.innerHTML+="  
>jWebSocket 连接已建立 " ;
    },
    OnMessage: function( aEvent, aToken ) {
        msg.innerHTML+="  
>jWebSocket \" " +aToken.type +
        "\" 令牌收到, 消息字符串为: \" " + aEvent.data + "\" " ;
    },
    OnClose: function( aEvent ) {
        msg.innerHTML+="  
>jWebSocket 连接被关闭. " ;
        document.getElementById("btnbroadcastText").disabled="disabled";
        document.getElementById("btnDisConnect").disabled="disabled";
        document.getElementById("btnTestPlugIn").disabled="disabled";
    }
});
msg.innerHTML+="  
>jWebSocketClient.resultToString(lRes);
if(lRes.code==0)
{
    document.getElementById("btnbroadcastText").disabled="";
    document.getElementById("btnDisConnect").disabled="";
    document.getElementById("btnTestPlugIn").disabled="";
}
}

```



```

function btnbroadcastText_click()
{
    var sendMsg=document.getElementById("sendMsg").value;
    var msg=document.getElementById("msg");
    msg.innerHTML+="  
> 广播消息: \""+sendMsg+"\"...";
    var lRes = jWebSocketClient.broadcastText("",sendMsg);
    if(lRes.code!=0)
        msg.innerHTML=jWebSocketClient.resultToString( lRes );
    document.getElementById("sendMsg").value="";
}
function btnDisConnect_click()
{
    if(jWebSocketClient)
    {
        var msg=document.getElementById("msg");
        msg.innerHTML+="  
> 用户 \""+userName+"\" 关闭连接 ";
        var lRes=jWebSocketClient.close();
        msg.innerHTML+="  
> "+jWebSocketClient.resultToString( lRes );
        if(lRes.code==0)
        {
            document.getElementById("btnbroadcastText").disabled="disabled";
            document.getElementById("btnDisConnect").disabled="disabled";
            document.getElementById("btnTestPlugIn").disabled="disabled";
        }
    }
}
function btnTestPlugIn_click()
{
    var msg=document.getElementById("msg");
    msg.innerHTML+="  
> 通过 WebSockets 获取服务器的系统时间 ... " ;
    var lRes = jWebSocketClient.requestServerTime();
    // 发生错误时显示错误消息
    if( lRes.code != 0 )
        msg.innerHTML+="  
> "+jWebSocketClient.resultToString(lRes);
}
function getServerTimeCallback( aToken ) {
    msg.innerHTML+="  
> 服务器的系统时间: " + aToken.time ;
}
function window_onunload()
{
    if(jWebSocketClient)
    {
        jWebSocketClient.close({timeout:3000});
    }
}
</script>

```

在打开页面时所调用的 window\_onload 函数中，OnSamplesServerTime 事件被触发时（当客户端接收到服务器端返回的对插件中定义的 requestServerTime 类型的令牌的响应令牌

时触发该事件)所执行的回调函数为 `getServerTimeCallback` 函数。

`getServerTimeCallback` 函数接受一个传入参数 `aToken`, 该参数代表服务器端对 `requestServerTime` 类型的令牌所返回的响应令牌。在函数中指定当客户端接收到该响应令牌时在通信消息显示区域中显示文字: “服务器的系统时间” + 令牌的 `time` 字段的值 (表示服务器端接收到 `requestServerTime` 类型的令牌时的系统时间)。

用户单击测试插件按钮时调用 `btnTestPlugIn_click` 函数, 该函数先将页面上的通信消息显示区域所使用的 `div` 元素赋值给变量 `msg`, 然后在通信消息显示区域中显示文字 “通过 WebSockets 获取服务器的系统时间…”, 随后调用客户端插件中的 `requestServerTime` 函数来获取服务器端的系统时间, 如果函数的返回值为 `-1`, 则表示获取失败, 在通信消息显示区域中显示错误信息。

最后来看一下使用了插件的页面在浏览器中的显示效果。用户连接 jWebSocket 服务器后单击测试插件按钮, 页面显示效果如图 10-17 所示。

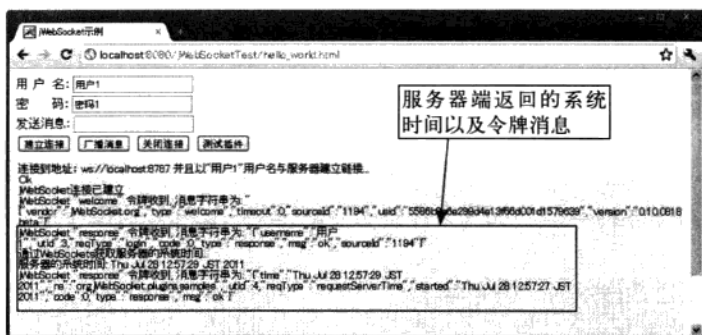


图 10-17 客户端通过使用插件来获取服务器的系统时间

## 10.6 jWebSocket 中的通道

本节将对 jWebSocket 中通道的基本知识进行详细阐述, 首先解释一下什么是 jWebSocket 中的通道。

jWebSocket 中的通道是一种高效的通信模式, 它允许多个应用程序或一个应用程序中的多个模块在不同的逻辑通道里在彼此不受影响的前提下互相交互。

jWebSocket 中的通道插件提供以下几种通道。

### □ 公有通道

所有的客户端都可以订阅并侦听一个公有通道, 并在这个通道上发布数据。与私有通道不同的是, 客户端可以通过 `getChannels` 请求获取所有公有通道的列表。可以通过访问 `key` 与私密 `key` 将公有通道保护起来。如果一个通道受到保护, 那么客户端必须提交正确的访问 `key` 才能订阅该通道。如果一个通道被设置了私密 `key`, 那么客户端需要提交正确的私密 `key`

才能在该通道上发布数据。

❑ 私有通道

私有通道用于在两个或多个客户端之间进行秘密通信。当使用 `getChannels` 请求时，这些通道不会出现在所获取的通道列表之中。另外，也必须通过访问 `key` 与私密 `key` 保护私有通道。订阅私有通道时，客户端必须知道通道 `id` 并提交正确的访问 `key`。只有提交了被授予的私密 `key` 才能在一个私有通道上发布消息。

❑ 系统通道

系统通道是保留给服务器使用的通道，只有服务器才能创建该通道（可以在 `jWebSocket.xml` 配置文件中配置），客户端不能更新与移除系统通道。

`jWebSocket.xml` 配置文件对通道的配置如下所示。

```
<setting key="channel:channelA" type="json">
{
  owner: "root",
  name: "Channel A",
  isPrivate: false,
  isSystem: true,
  accessKey: "Access",
  secretKey: "Secret"
}
</setting>
```

注意，目前 `jWebSocket.xml` 配置文件中所有的 JSON 设置必须被写在一行中（可以使用逗号进行分隔）。

接下来看一下通道使用的一些令牌。

❑ subscribe 令牌

客户端通过发送 `subscribe` 令牌来订阅通道。如果一个通道需要提交一个访问 `key` 才能被订阅（尤其是私有通道），那么必须在令牌的 `accessKey` 字段中填入正确的访问 `key`；如果不需要，可以不填入 `accessKey` 字段。如果通道不存在或访问 `key` 不正确，订阅通道的请求将被拒绝。如果成功订阅通道，客户端将接收到这个通道上的消息，直到客户端退订为止。

`subscribe` 令牌中所包含的字段及其说明如表 10-19 所示。

表 10-19 subscribe 令牌中所包含的字段及其说明

字段	数据类型	说明
type	String	始终为“subscribe”
ns	String	命名空间为“org.jWebSocket.plugins.channels”
channel	String	唯一通道标识符
accessKey	String	订阅通道时使用的访问 key（如果通道被配置为需要提交访问 key）
secretKey	String	订阅通道时不需要填写

### ❑ unsubscribe 令牌

客户端通过发送 unsubscribe 令牌来退订通道。如果通道不存在，退订通道的请求将被拒绝。如果成功退订通道，客户端将再也接收不到这个通道上的任何消息。退订通道时客户端不需要提交访问 key 和私密 key。

unsubscribe 令牌中所包含的字段及其说明如表 10-20 所示。

表 10-20 unsubscribe 令牌中所包含的字段及其说明

字段	数据类型	说明
type	String	始终为“unsubscribe”
ns	String	命名空间为“org.jWebSocket.plugins.channels”
channel	String	唯一通道标识符

### ❑ getChannels 令牌

客户端发送 getChannels 令牌后服务器端返回所有的公有通道列表。私有通道将不会出现在该列表中。基于安全原因，通道的拥有者将不会被返回。

getChannels 令牌中所包含的字段及其说明如表 10-21 所示。

表 10-21 getChannels 令牌中所包含的字段及其说明

字段	数据类型	说明
type	String	始终为“getChannels”
ns	String	命名空间为“org.jWebSocket.plugins.channels”

客户端发送 getChannels 令牌后，服务器端返回的令牌中所包含的字段及其说明如表 10-22 所示。

表 10-22 服务器端返回的令牌中所包含的字段及其说明

字段	数据类型	说明
channels	Array	通道对象的数组。通道对象包括以下几个属性： id (String 类型，表示通道 id) name (String 类型，表示通道名称) isSystem (Boolean 类型，表示是否为系统通道) isPrivate (Boolean 类型，表示是否为私有通道)

### ❑ getSubscriptions 令牌

客户端发送 getSubscriptions 令牌后服务器端返回该客户端所订阅的通道列表。私有通道不会出现在该列表中。基于安全原因，通道的拥有者将不会被返回。

getSubscriptions 令牌中所包含的字段及其说明如表 10-23 所示。

表 10-23 getSubscriptions 令牌中所包含的字段及其说明

字段	数据类型	说明
type	String	始终为 “getSubscriptions”
ns	String	命名空间为 “org.jWebSocket.plugins.channels”

客户端发送 getSubscriptions 令牌后，服务器端返回的令牌中所包含的字段及其说明如表 10-24 所示。

表 10-24 服务器端返回的令牌中所包含的字段及其说明

字段	数据类型	说明
chanel	Array	通道对象的数组。通道对象包括以下几个属性： id (String 类型，表示通道 id) name (String 类型，表示通道名称) isSystem (Boolean 类型，表示是否为系统通道) isPrivate (Boolean 类型，表示是否为私有通道)

❑ getSubscribers 令牌

客户端发送 getSubscribers 令牌后，服务器端返回该客户端指定通道的所有订阅者列表。getSubscribers 令牌中所包含的字段及其说明如表 10-25 所示。

表 10-25 getSubscribers 令牌中所包含的字段及其说明

字段	数据类型	说明
type	String	始终为 “getSubscribers”
ns	String	命名空间为 “org.jWebSocket.plugins.channels”
channel	String	指定通道的唯一通道标识符
accessKey	String	指定通道的访问 key（如果通道被配置为需要提交访问 key）

客户端发送 getSubscribers 令牌后，服务器端返回的令牌中所包含的字段及其说明如表 10-26 所示。

表 10-26 服务器端返回的令牌中所包含的字段及其说明

字段	数据类型	说明
channel	String	指定通道的唯一通道标识符
subscribers	Array	指定由通道的所有订阅者组成的数组（每一个数组项为 String 类型的客户端 id）

❑ authorize 令牌

客户端通过发送 authorize 令牌来请求获得在一个通道上发布数据的权利。客户端只有被授予了发布数据的权利才能在通道上发布数据。

authorize 令牌中所包含的字段及其说明如表 10-27 所示。

表 10-27 authorize 令牌中所包含的字段及其说明

字段	数据类型	说明
type	String	始终为“authorize”
ns	String	命名空间为“org.jWebSocket.plugins.channels”
channel	String	请求获得权利的通道的唯一通道标识符
accessKey	String	在通道上发布数据时使用的访问 key(如果通道被配置为需要提交访问 key)
secretKey	String	在通道上发布数据时使用的私密 key(如果通道被配置需要提交私密 key)

### □ publish 令牌

客户端通过发布 publish 令牌在通道上发布数据。发布数据之前客户端先要取得发布数据的权利。

publish 令牌中所包含的字段及其说明如表 10-28 所示。

表 10-28 publish 令牌中所包含的字段及其说明

字段	数据类型	说明
type	String	始终为“publish”
ns	String	命名空间为“org.jWebSocket.plugins.channels”
channel	String	通道的唯一通道标识符
Data	String	要在通道上发布的数据

### □ createChannel 令牌

客户端通过发送 createChannel 令牌在服务器上创建通道。如果已经存在具有相同 id 的通道,则创建通道的请求将被拒绝。创建私有通道时客户端必须提供一个访问 key,否则创建通道的请求将被拒绝。创建公共通道时可以不提供访问 key。

createChannel 令牌中所包含的字段及其说明如表 10-29 所示。

表 10-29 createChannel 令牌中所包含的字段及其说明

字段	数据类型	说明
type	String	始终为“createChannel”
ns	String	命名空间为“org.jWebSocket.plugins.channels”
channel	String	创建的通道的唯一通道标识符
accessKey	String	创建的通道的访问 key(创建私有通道时必须填入访问 key)
secretKey	String	创建的通道的私密 key(创建私有通道时必须填入私密 key)
isPrivate	Boolean	创建的通道是否为私有通道
isSystem	Boolean	创建的通道是否为系统通道
owner	String	通道拥有者的用户 id。只有通道拥有者可以移除通道。如果不在该字段中填入拥有者,则客户端的当前已登录到 jWebSocket 服务器的用户 id 自动成为通道的拥有者

### ❑ removeChannel 令牌

客户端通过发送 removeChannel 令牌在服务器上移除通道。只有通道的拥有者可以移除通道。客户端在提交了正确的通道访问 key 或私密 key 后才能移除通道，否则请求将被拒绝。

removeChannel 令牌中所包含的字段及其说明如表 10-30 所示。

表 10-30 removeChannel 令牌中所包含的字段及其说明

字段	数据类型	说明
type	String	始终为 “removeChannel”
ns	String	命名空间为 “org.jWebSocket.plugins.channels”
channel	String	通道的唯一通道标识符
accessKey	String	移除通道时所使用的访问 key（如果通道被配置为需要提交访问 key）
secretKey	String	移除通道时所使用的私密 key（如果通道被配置为需要提交私密 key）
owner	String	通道拥有者（创建通道的用户）的用户 id。如果不在该字段中填入拥有者，则客户端的当前已登录到 jWebSocket 服务器的用户 id 自动作为该字段值来使用

## 10.7 案例 28：利用 jWebSocket 服务器创建简单聊天室

### 10.7.1 案例概述

本节通过一个利用 jWebSocket 服务器创建简单聊天室的案例来进一步展示如何创建使用 jWebSocket 服务器进行通信的客户端页面。此页面中有一个聊天室，用户可以在此页面中输入用户名后单击登录按钮登录聊天服务器，然后与其他已登录聊天服务器的用户进行文字聊天。页面中还显示一个用户列表，当用户登录或退出聊天室时随时更新用户列表，显示当前登录到聊天室中的所有用户的用户名 + “@” + 该用户的客户端 id。

### 10.7.2 页面显示效果

首先来看一下在浏览器中打开案例页面时的显示效果，如图 10-18 所示（本案例使用的浏览器为 Google Chrome 10）。

用户可以在用户名文本框中输入登录聊天服务器的用户名，如图 10-19 所示。

用户在用户名文本框中输入登录聊天服务器的用户名后单击登录按钮，页面中显示该用户进入聊天室的系统消息以及聊天室对用户的欢迎消息，同时该用户的用户名被追加到用户列表中，如图 10-20 所示。

其他用户进入聊天室时本页面上也会显示该用户进入聊天室的系统消息及聊天室对用户的欢迎消息，同时相应的用户名被追加到用户列表中。

测试时可以重新打开一个 Google Chrome 浏览器并访问本页面，然后输入一个用户名并用单击登录按钮，打开的案例页面上将显示该用户进入聊天室的系统消息以及聊天室对用户

的欢迎消息，同时该用户的用户名被追加到用户列表中，如图 10-21 所示。

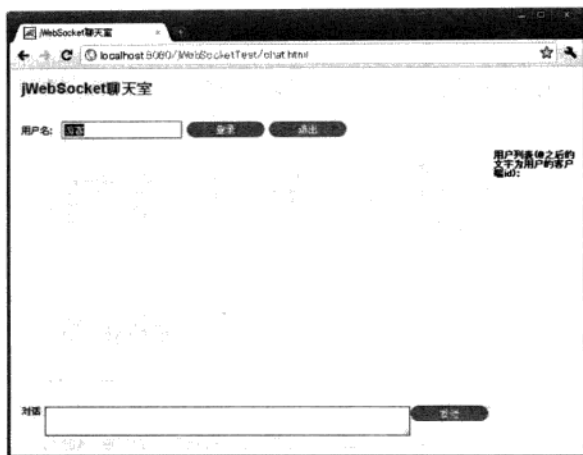


图 10-18 在浏览器中打开案例页面时的显示效果

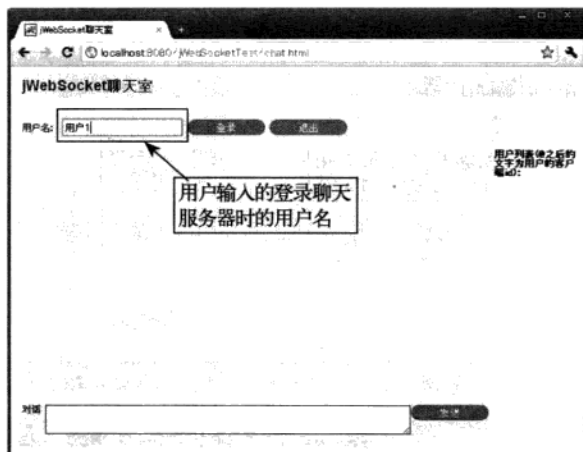


图 10-19 用户在用户名文本框中输入登录聊天服务器的用户名

用户在对话文本框中输入内容，然后单击发送按钮，对话文本框中的内容会被广播给其他所有登录到 jWebSocket 聊天服务器的用户，页面上的聊天消息区域中也会显示用户在对话文本框中输入的内容，如图 10-22 所示。

当其他用户广播消息时，本用户也会接收到其他用户广播的消息。测试时在第二个打开的 Google Chrome 浏览器的对话文本框中输入内容，然后单击发送按钮，返回到打开的第一个 Google Chrome 浏览器，此浏览器中显示在第二个 Google Chrome 浏览器中登录到 jWebSocket 聊天服务器的用户所发送的内容，如图 10-23 所示。



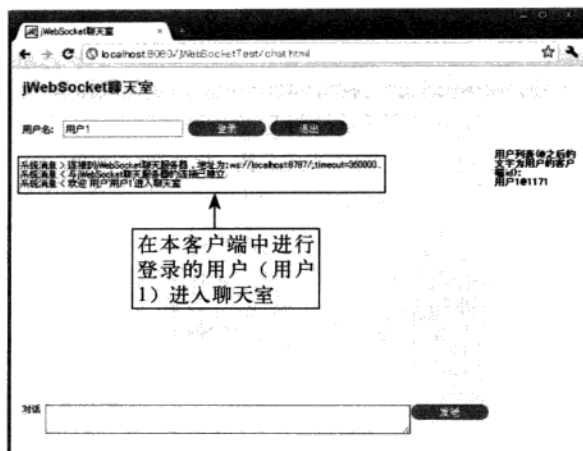


图 10-20 用户进入聊天室时页面上显示系统消息与欢迎消息

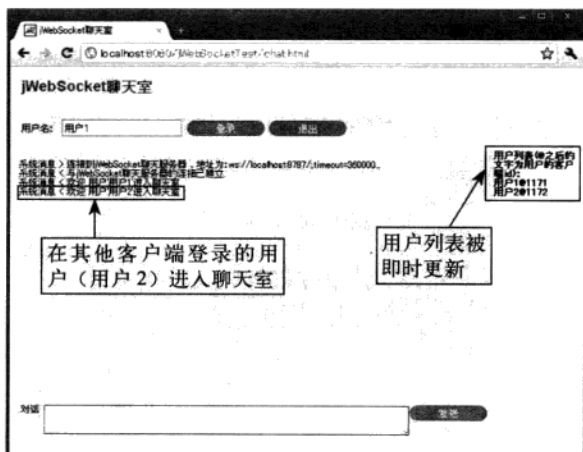


图 10-21 其他用户进入聊天室时本页面中会显示相关消息

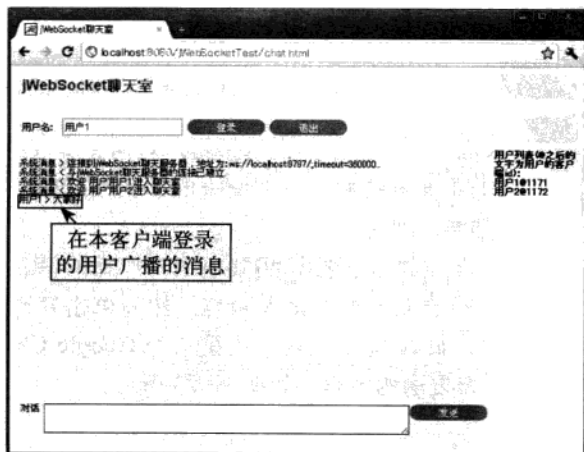


图 10-22 用户向其他用户广播聊天消息

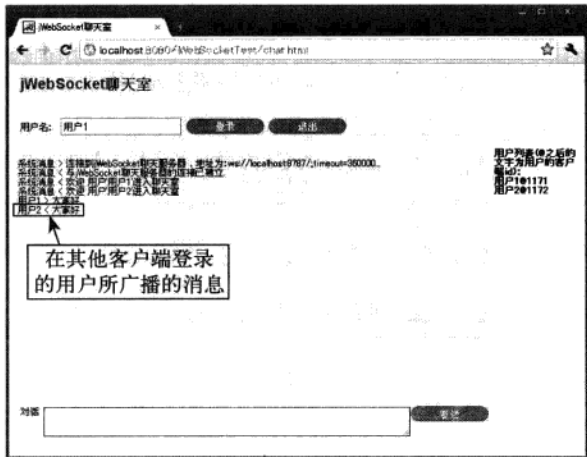


图 10-23 本浏览器能接收到在其他客户端登录的用户所广播的消息

10.7.3 代码剖析

1. HTML 页面代码以及 CSS 样式代码部分

接下来对本案例中所使用的代码进行详细分析。首先分析本案例的 HTML 页面代码以及 CSS 样式代码部分。

案例页面中使用的元素如表 10-31 所示。

表 10-31 案例页面中使用的元素

元素名称	元素类型	显示文字	说明
“jWebSocket 聊天室” 文字	h1	jWebSocket 聊天室	
“用户名:” 文字	页面文字	用户名:	
用户名文本框	input type="text"		页面打开时该文本框中的初始值为“游客”
登录按钮	input type="button"	登录	单击登录按钮时客户端以用户在用户名文本框中输入的用户名与空白密码登录到 jWebSocket 聊天服务器
退出按钮	input type="button"	退出	单击退出按钮时当前客户端与 jWebSocket 聊天服务器断开连接。页面打开时为无效状态, 用户登录到 jWebSocket 聊天服务器后变为有效状态, 客户端与服务端之间的连接被关闭时变为无效状态
聊天消息与系统消息显示区域	div		显示多个客户端与 jWebSocket 聊天服务进行交互的系统消息及多个客户端用户广播的聊天信息
用户列表显示区域	div		实时显示登录到 jWebSocket 聊天服务器的用户列表。页面打开时的初始文字为“用户列表 (@ 之后的文字为用户的客户端 id):”
“对话” 文字	页面文字	对话	
对话文本框	textarea		用户在其中输入聊天内容

(续)

元素名称	元素类型	显示文字	说明
发送按钮	input type="button"	发送	单击发送按钮时用户在对话文本框中输入的内容被广播给其他所有登录到 jWebSocket 聊天服务器的用户。页面打开时为无效状态, 用户登录到 jWebSocket 聊天服务器后变为有效状态, 客户端与服务器端之间的连接被关闭时变为无效状态

本案例的完整 HTML 页面代码与 CSS 样式代码如代码清单 10-10 所示。

代码清单 10-10 本案例的完整 HTML 页面代码与 CSS 样式代码

```

<!DOCTYPE html>
<html>
<head>
<title>jWebSocket 聊天室 </title>
<style>
h1 {
    font-family: Arial, Helvetica, sans-serif;
    font-weight: bold;
    font-size: 14pt;
    color: #006bb5;
    background-color: #f0f0f0;
    -moz-border-radius: 5px;
    -webkit-border-radius: 5px;
    border: 1px solid #f0f0f0;
    padding: 5px 5px 5px 5px;
    margin: 0px 0px 18px 0px;
}
div[id^=divContainer]{
    border: 0px;
    margin:10px 0px 10px 0px;
    padding: 3px;
    background-color: #f0f0f0;
    -moz-border-radius: 5px;
    -webkit-border-radius: 5px;
}
div#divLeft{
    width:85%;
    background-color: #f0f0f0;
    float:left;
}
div#divRight{
    width:15%;
    background-color:white;
    float:right;
    font-weight:bold;
    font-size:12px;
}

```

```

div#divchat{
    border: 0px;
    margin:10px 0px 10px 0px;
    padding: 3px;
    background-color: #f0f0f0;
    -moz-border-radius: 5px;
    -webkit-border-radius: 5px;
    position:relative;
    height:300px;
    overflow:auto;
    font-size: 12px;
}
table#tbDlg {
    font-family: Verdana, Helvetica, sans-serif;
    font-weight: normal;
    font-size: 12px;
    background-color: #f0f0f0;
}
tr#trDlg, td#tdDlg {
    background-color: #f0f0f0;
    font-size: 10px;
}
textarea {
    font-family: inherit;
    font-size: 10pt;
    border: 1px solid #444;
    background-color: white;
    width:100%;
}
input[type="button"] {
    font-family: inherit;
    border: 1px solid #808080;
    -moz-border-radius: 10px;
    -webkit-border-radius: 10px;
    margin: 1px;
    color: white;
    background-color: #81a0b5;
    width: 100px;
}
input[type="button"]:hover {
    margin: 1px;
    background-color: #006bb5;
}
input[type="button"]:active {
    margin: 1px;
    font-weight: bold;
    background-color: #006bb5;
}
input[type="button"]:focus {
    margin: 0px;
}

```

```

        font-weight: bold;
        background-color: #006bb5;
    }
</style>
</head>
<script src="jWebSocket.js" type="text/javascript"></script>
<script type="text/javascript">
... 脚本代码稍后介绍 ...
</script>
<body onload="window.onload()" onunload="window.onunload()">
<h1>jWebSocket 聊天室 </h1>
<div id="divContainer1">
    <table id="tbDlg" border="0" cellpadding="3" cellspacing="0" width="100%">
        <tr id="trDlg">
            <td id="tdDlg" width="5">
                用户名: &nbsp;
                <input id="tbxUsername" type="text" value=" 游客 " size="20">
                <input id="btnLogin" type="button" value=" 登录 "
                onclick="btnLogin_onclick();">
                <input id="btnLogout" type="button" value=" 退出 " disabled
                onclick="btnLogout_onclick();">
            </td>
        </tr>
    </table>
</div>
<div id="divLeft">
    <div id="divchat"></div>
    <div id="divContainer3">
        <table id="tbDlg" border="0" cellpadding="3" cellspacing="0"
        width="100%">
            <tr id="trDlg">
                <td valign="top" id="tdDlg" nowrap> 对话 </td>
                <td valign="top" id="tdDlg">
                    <textarea id="tbxMsg" cols="255" rows="2"
                    style="width:100%"></textarea>
                </td>
                <td valign="top" id="tdDlg">
                    <input id="btnSend" type="button" value=" 发送 " disabled
                    onclick="btnSend_onclick();" >
                </td>
            </tr>
        </table>
    </div>
</div>
<div id="divRight">
    用户列表 (@之后的文字为用户的客户端 id):
</div>
</body>
</html>

```

---

## 2. KeepAlive 功能

在对本案例中使用的 JavaScript 脚本代码进行详细分析之前，先简单介绍一下这段脚本代码中所使用的一个重要功能——KeepAlive 功能的含义及其使用方法。

在使用 jWebSocket 框架进行 Socket 通信的时候，当客户端处于非活动状态（客户端不向服务器端发出任何请求）一段时间且该时间超出指定的 timeout 时间值后，服务器端将中止会话，将客户端与服务器端之间的连接关闭。因为服务器端不能主动对客户端进行操作，所以通过指定超时时间来管理会话与连接是一种必需的管理机制。这样客户端可以通过主动发送 close 令牌来向服务器端请求关闭客户端与服务器端的连接，服务器端也可以在指定的超时时间过去之后将其与一些由于网络原因而与服务器端意外断开连接（没有向服务器端发出关闭连接请求而被意外中断连接）的客户端之间的连接关闭，将被这些客户端占用的端口释放。如果没有这种超时管理机制，服务器端的端口将很快被用尽（因为得不到释放）。

超时管理机制是以客户端是否在指定时间范围内与服务器端进行交互操作为依据进行管理的，如果超出超时时间而客户端没有向服务器端发出任何请求，服务器端就结束会话，关闭连接。在某些特殊场合下（例如，在网页中展示较长篇幅的文章或其他流数据时），用户在较长时间内不再向服务器端发出任何请求，只是处于对文章或流数据进行阅读的状态中，这时尽管超出了超时限制，用户还是希望服务器端保持与客户端的连接。在这种情况下，可以让客户端每隔一段时间向服务器端自动发送一个 ping 令牌以声明自己处于活动状态（没有因为网络故障而意外断开连接），以确保服务器端不会结束会话，不会断开连接。服务器端也可以向客户端返回一个响应令牌，客户端根据这个响应令牌来确认服务器端与自己处于连接状态。客户端每隔一段时间自动发送 ping 令牌来声明自己处于活动状态的功能就叫 KeepAlive 功能。打开 KeepAlive 功能的代码如下所示。

```
jWebSocketClient.startKeepAlive(options);
```

在上面这句代码中，jWebSocketClient 为一个 jWebSocketJSONClient 类的对象，通过该语句来启动一个 KeepAlive 计时器。该计时器控制客户端每隔一段时间自动向服务器端发送一个 ping 令牌。如果执行该语句时 KeepAlive 计时器已经启动，则之前启动的 KeepAlive 计时器被自动停止，重新启动一个新的 KeepAlive 计时器，并且通过 options 参数对该计时器进行初始化工作。

options 参数中保存了几个可选参数，通过这些参数可以初始化 KeepAlive 计时器。这些参数如下所示。

- ❑ options.interval：指定计时器的时间间隔，以毫秒为单位，参数值为整数类型的毫秒数。
- ❑ options.echo：指定服务器端是否需要向客户端返回响应令牌。参数值为布尔类型（True 或 False）。
- ❑ options.immediate：指定执行该语句后客户端是否立即发送第一个 ping 令牌（而无需等待计时器的通知）。

停止 KeepAlive 功能的代码如下所示。

```
jWebSocketClient.stopKeepAlive();
```

执行完该语句后, KeepAlive 计时器停止工作, 客户端不再向服务器端发送 ping 令牌。

### 3. JavaScript 脚本代码部分

接下来看一下本案例中使用的 JavaScript 脚本代码部分, 如代码清单 10-11 所示。

代码清单 10-11 本案例中使用的 JavaScript 脚本代码

---

```
<script src="jWebSocket.js" type="text/javascript"></script>
<script type="text/javascript">
var jWebSocketClient;
var divChat,tbxUsername,tbxMsg,userName;
var IN=0,OUT=1;
var SYS=" 系统消息 ";
function window_onload()
{
    divChat=document.getElementById("divchat");
    tbxUsername=document.getElementById("tbxUsername");
    tbxMsg=document.getElementById("tbxMsg");
    if(jws.browserSupportsWebSockets())
    {
        jWebSocketClient = new jws.jWebSocketJSONClient();
        tbxUsername.focus();
        tbxUsername.select();
    }
    else
    {
        document.getElementById("btnSend").disabled="disabled";
        document.getElementById("btnLogin").disabled="disabled";
        document.getElementById("btnLogout").disabled="disabled";
        var lMsg = jws.MSG_WS_NOT_SUPPORTED;
        alert( lMsg );
        log(SYS, IN, lMsg );
    }
}
function log(username,event,string ) {
    var lFlag;
    if(event==IN)
        lFlag = "<";
    else
        lFlag = ">";
    if(!username)
        username = jWebSocketClient.getUsername();
    // 如果用户没有登录, 则设置username 为默认用户名
    if( !username )
        username = " 游客 ";
    divChat.innerHTML+=username + " " +lFlag + " " +string + "<br>";
    if( divChat.scrollHeight > divChat.clientHeight )
        divChat.scrollTop = divChat.scrollHeight - divChat.clientHeight;
```

```

}
function btnLogin_onclick()
{
    var lURL = jws.JWS_SERVER_URL + "/; ,timeout=360000";
    var clientArray;
    if(tbxUsername.value.trim()=="")
    {
        alert(" 请输入用户名 ");
        return;
    }
    log(SYS,OUT," 连接到 jWebSocket 聊天服务器 , 地址为: " + lURL + "..." );
    var lRes=jWebSocketClient.logon(lURL,tbxUsername.value, "", {
        OnOpen: function(aEvent){
            log(SYS,IN," 与 jWebSocket 聊天服务器的连接已建立.");
            // 以下为可选代码, 是否打开 KeepAlive 功能
            /*var options={};
            options.immediate=false;
            options.interval = 30000;
            jWebSocketClient.startKeepAlive(options);*/
        },
        OnMessage: function( aEvent, aToken ) {
            if(aToken)
            {
                if(aToken.type == "response")
                {
                    if(aToken.reqType == "login")
                    {
                        if( aToken.code == 0 )
                        {
                            log(SYS, IN, " 欢迎用户 '" + aToken.username+
                                "' 进入聊天室 " );
                            jWebSocketClient.getAuthClients({pool: null});
                        }
                        else
                            log(SYS,IN," 登录失败, 错误消息为: "+aToken.msg );
                    }
                }
                else if(aToken.reqType == "getClients")
                {
                    var divRight=document.getElementById("divRight");
                    divRight.innerHTML=" 用户列表 (@ 之后的文字为用户的客户端 id): ";
                    for(var i=0;i<aToken.clients.length;i++)
                    {
                        divRight.innerHTML+="<br/>" + aToken.clients[i];
                    }
                }
            }
            else if(aToken.type == "goodBye")
                log(SYS,IN,"jWebSocket 聊天服务器 断开与客户端的连接 (原因: " +
                    aToken.reason + ")!" );
            else if(aToken.type == "broadcast")

```



```

        {
            if (aToken.data)
                log( aToken.sender, IN, aToken.data);
        }
        else if (aToken.type == "event")
        {
            jWebSocketClient.getAuthClients({pool: null});
            var data=JSON.parse(aEvent.data);
            if (data.name=="login")
            {
                log(SYS, IN, " 欢迎 用户 '" + data.username+"' 进入聊天室 " );
            }
            if (data.name=="logout")
            {
                log(SYS, IN, " 用户 '" + data.username+"' 退出聊天室 " );
            }
        }
    }
},
OnClose:function(aEvent){
    log(SYS,IN," 与 jWebSocket 聊天服务器 的连接已关闭 .");
    document.getElementById("btnSend").disabled="disabled";
    document.getElementById("btnLogout").disabled="disabled";
    // 以下是可选代码, 当 KeepAlive 功能被打开时将其关闭
    //jWebSocketClient.stopKeepAlive();
}
});
if(lRes.code==0)
{
    userName=tbxUsername.value;
    document.getElementById("btnSend").disabled="";
    document.getElementById("btnLogout").disabled="";
}
}
function btnSend_onclick()
{
    var msg = tbxMsg.value;
    if(msg.length > 0)
    {
        log(userName,OUT,msg);
        var lRes = jWebSocketClient.broadcastText("",msg);
        if(lRes.code!=0)
            log(SYS,OUT,lRes.msg);
        tbxMsg.value="";
    }
}
function btnLogout_onclick()
{
    var lRes = jWebSocketClient.close();
    log(SYS, OUT, " 用户 "+userName+" 退出聊天室: "+ lRes.msg );
}

```

```

if (lRes.code==0)
{
    document.getElementById("btnSend").disabled="disabled";
    document.getElementById("btnLogout").disabled="disabled";
}
}
function window_onunload()
{
    if (document.getElementById("btnLogout").disabled=="")
        jWebSocketClient.close();
}
</script>

```

在 JavaScript 脚本代码的开头，定义了几个此脚本中用到的全局变量，它们的含义如下。

- ❑ jWebSocketClient：代表 jWebSocket 中使用的一个 jWebSocketJSONClient 类的对象。
- ❑ divChat：代表页面中的聊天消息与系统消息显示区域。
- ❑ tbxUsername：代表页面中的用户名文本框。
- ❑ tbxMsg：代表页面中的对话文本框。
- ❑ userName：代表用户登录到 jWebSocket 聊天服务器时所使用的用户名。
- ❑ IN 与 OUT：用来标识消息是本客户端发出的消息还是本客户端接收到的消息。在聊天消息与系统消息显示区域中的发出消息前加上“>”前缀文字，用来突出显示该条消息为本客户端发出的消息；在聊天消息与系统消息显示区域中的接收消息前加上“<”前缀文字，用来突出显示该条消息为本客户端接收到的消息。
- ❑ 变量 SYS 相当于一个常量，变量值始终为“系统消息”，用来在聊天消息与系统消息显示区域中显示的系统消息前加上“系统消息”文字。

接下来对本脚本代码中用到的几个函数进行详细分析。

#### ❑ window\_onload 函数

打开案例页面时调用 window\_onload 函数，该函数首先获取页面上的聊天消息与系统消息显示区域并赋值给全局变量 divChat，获取页面上的用户名文本框并赋值给全局变量 tbxUsername，获取页面上的对话文本框并赋值给全局变量 tbxMsg。然后检测浏览器是否支持 WebSocket 通信，如果支持则创建一个 jWebSocketJSONClient 对象并赋值给全局变量 jWebSocketClient，同时将光标焦点设置在用户名文本框中；如果不支持则将页面上的所有按钮设定为无效状态，然后弹出相应的错误提示信息并调用 log 函数将错误提示信息显示在聊天消息与系统消息显示区域中。

#### ❑ log 函数

log 函数用于将系统消息（客户端与服务器端之间建立连接和断开连接等交互信息）或用户广播的消息显示在系统消息或聊天消息显示区域中。该函数接收 3 个传入参数，第一个传入参数 username 表示用户登录到 jWebSocket 聊天服务器时所使用的用户名；第二个传入参数 event 用来区分该消息是客户端发出的消息还是接收到的消息，参数值为 IN 或 OUT（在

脚本代码的开头处定义 IN=0, OUT=1), IN 表示消息为客户端接收到的消息, OUT 表示消息为客户端发出的消息; 第三个传入参数 string 表示显示在聊天消息与系统消息显示区域中的消息字符串。

在函数内部首先定义一个代表前缀字符串的变量 IFlag。如果传入参数 event=IN, 则设置 IFlag 变量值为 “<”, 表示接下来在聊天消息与系统消息显示区域中显示的消息为客户端接收到的消息; 否则设置 IFlag 变量值为 “>”, 表示接下来在聊天消息与系统消息显示区域中显示的消息为客户端主动发出的消息。接下来判断传入参数 username (代表登录到 jWebSocket 聊天服务器时所使用的用户名) 是否已被赋值, 如果未赋值则调用 jWebSocketJSONClient 对象的 getUsername 方法获取本客户端登录到 jWebSocket 聊天服务器时所使用的用户名, 如果客户端尚未登录到 jWebSocket 聊天服务器, 则将用户名设定为 “游客”。然后在聊天消息与系统消息显示区域中显示传入参数 string 所代表的消息字符串, 显示文字为: 用户名 + “ ” + IFlag 变量值 (“<” 或 “>”) + “ ” + 传入参数 string 所代表的消息字符串。最后判断聊天消息与系统消息显示区域中的所有文字高度是否已超出在 HTML 代码中指定的高度, 如果超出则在聊天消息与系统消息显示区域中显示滚动条。

#### □ btnLogin\_onclick 函数

用户单击登录按钮时调用 btnLogin\_onclick 函数。该函数首先通过 jws 命名空间中的 JWS\_SERVER\_URL 变量获取 jWebSocket 聊天服务器所在地的 URL 地址并赋值给变量 IURL, 同时设置客户端登录到聊天服务器后该客户端在聊天服务器中的超时时间为 360 秒 (6 分钟)。如果在指定时间范围内客户端没有与服务器端进行交互, 服务器将关闭与客户端的连接。接着定义变量 clientArray 来存放所有登录到 jWebSocket 聊天服务器的用户的用户列表。接下来判断用户是否在用户名文本框中输入了登录 jWebSocket 聊天服务器时所使用的用户名, 如果未输入, 则弹出错误提示信息, 文字为 “请输入用户名”; 如果已经输入, 则调用 log 函数在聊天消息与系统消息显示区域中显示一条系统消息, 用户名为 SYS 常量值 (值为 “系统消息”, 在聊天消息与系统消息显示区域中该条消息的开头显示 “系统消息”), event 参数值为 OUT (在聊天消息与系统消息显示区域中该条消息的 “系统消息” 文字后显示 “>”, 表示该条消息为客户端主动发出的消息), string 参数值所代表的消息字符串的文字为: “连接到 jWebSocket 聊天服务器, 地址为: ” + jWebSocket 聊天服务器所在地的 URL 地址 + “...”。

接下来使用 jWebSocketJSONClient 对象的 logon 方法连接并使用用户输入的用户名与空白密码登录到 jWebSocket 聊天服务器, 将登录结果赋值给变量 IRes, 同时指定连接建立后所执行的回调函数 OnOpen。在该回调函数中指定在聊天消息与系统消息显示区域中显示 “与 jWebSocket 聊天服务器的连接已建立。” 消息字符串, 用户名为 SYS (表示该条消息为系统消息), event 参数值为 IN (表示该条消息为客户端接收到的消息)。OnOpen 函数的结尾处为一段可选代码, 其作用是为客户端开启 KeepAlive 功能, 指定客户端立即发送第一个 ping 令牌, 并且每隔 3 秒自动发送一个 ping 令牌。

指定当客户端接收到服务器端发送到的令牌后所执行的回调函数 OnMessage。该回调函数首先判断传入参数 aToken (表示接收到的令牌) 是否存在, 如果令牌不存在则退出该回调

函数；如果令牌存在则首先判断令牌类型，并执行如下处理。

□ 令牌类型为 response 且令牌的 reqType 字段值为 login

令牌类型为 response 表示该令牌是服务器端在接收到客户端发出的令牌后所返回的令牌，令牌的 reqType 字段值为 login 表示客户端发出的请求为“登录到 jWebSocket 聊天服务器”。这时首先判断 aToken 参数所代表的令牌的 code 字段值是否为 0，code 字段值为 0 则表示用户成功登录 jWebSocket 聊天服务器，此时在聊天消息与系统消息显示区域中显示消息字符串，文字为：“欢迎 用户'” + 用户登录 jWebSocket 聊天服务器时所使用的用户名 + “' 进入聊天室”，用户名为 SYS，event 参数值为 IN。在聊天消息与系统消息显示区域中显示消息字符串后调用 jWebSocketJSONClient 对象的 getAuthClients 方法获取所有登录到 jWebSocket 聊天服务器的用户名，aToken 参数所代表的令牌的 code 字段值不为 0 时表示用户登录到 jWebSocket 聊天服务器的操作失败，这时在聊天消息与系统消息显示区域中显示消息字符串，文字为：“登录失败，错误消息为：” + 服务器端返回的系统错误消息。

□ 令牌类型为 response 且令牌的 reqType 字段值为 getClients

使用 jWebSocketJSONClient 对象的 getAuthClients 方法获取所有登录到 jWebSocket 聊天服务器的用户名时，服务器端返回的 reqType 字段值为 getClients 的 response 响应令牌。当客户端接收到该令牌时更新用户列表显示区域，在该区域中首先显示文字“用户列表 (@ 之后的文字为用户的客户端 id):”，然后显示所有登录到 jWebSocket 聊天服务器的用户名列表。

□ 令牌类型为 goodBye

当客户端主动关闭与服务器端之间的连接时，服务器端将向客户端发出类型为 goodBye 的响应令牌，这时在聊天消息与系统消息显示区域中显示消息字符串，文字为：“jWebSocket 聊天服务器断开与客户端的连接 (原因: ” + goodBye 响应令牌中 reason 字段值 (表示引起服务器与客户端之间连接被关闭的原因) + “)!”，用户名为 SYS，event 参数值为 IN。

□ 令牌类型为 broadcast

当其他客户端的用户向所有登录到 jWebSocket 聊天服务器的用户广播消息时，本客户端会接收到服务器转发的类型为 broadcast 的令牌，该令牌的数据中保存了该客户端所广播的消息字符串。这时脚本代码先判断该客户端所广播的消息字符串是否为空，如果不为空则在聊天消息与系统消息显示区域中显示该客户端所广播的消息字符串，用户名为该令牌的 sender 字段值，event 参数值为 IN。

□ 令牌类型为 event

当其他客户端的用户与 jWebSocket 聊天服务器进行连接或断开连接等交互操作时，本客户端会接收到类型为 event 的令牌，该令牌中包含了其他客户端与 jWebSocket 聊天服务器的操作信息。在脚本代码中指定当客户端接收到该令牌时首先调用 jWebSocketJSONClient 对象的 getAuthClients 方法获取所有登录到 jWebSocket 聊天服务器的用户名，然后实时更新用户列表，重新显示当前登录到 jWebSocket 聊天服务器的所有用户的用户名列表。然后通过 JSON 对象的 parse 方法将回调函数的传入参数 aEvent 的数据属性值解析为 JavaScript 对象，随后判断该对象的名字字段值 (代表其他客户端与 jWebSocket 聊天服务器之间进行交

互操作的操作种类)。如果该字段值为“login”，表示其他用户登录到 jWebSocket 聊天服务器，这时在聊天消息与系统消息显示区域中显示消息字符串，文字为：“欢迎 用户'” + 登录到 jWebSocket 聊天服务器的用户的用户名 + “' 进入聊天室”，用户名为 SYS，event 参数值为 IN。如果该字段值为“logout”，表示其他用户退出 jWebSocket 聊天服务器，这时在聊天消息与系统消息显示区域中显示消息字符串，文字为：“欢迎 用户'” + 退出 jWebSocket 聊天服务器的用户的用户名 + “' 退出聊天室”，用户名为 SYS，event 参数值为 IN。

在 onClose 回调函数中指定，当客户端与 jWebSocket 服务器端之间的连接被服务器端显式关闭时，在聊天消息与系统消息显示区域中显示“与 jWebSocket 聊天服务器的连接已关闭。”文字，用户名为 SYS，event 参数值为 IN，消息显示完毕后将退出按钮与发送按钮设为无效状态。onClose 回调函数的结尾为一段可选代码，其作用是，当 KeepAlive 功能被打开时将其关闭，客户端不再自动发送 ping 令牌。

在 btnLogin\_onclick 函数的最后，判断用户是否成功登录到 jWebSocket 聊天服务器，如果登录成功，则将用户在用户名文本框中输入的用户名清除，同时将退出按钮与发送按钮设定为有效状态。

#### ❑ btnSend\_onclick 函数

用户在对话文本框中输入聊天内容，在单击发送按钮时调用 btnSend\_onclick 函数。该函数首先将用户在对话文本框中输入的聊天内容赋值给变量 msg，然后判断聊天内容的长度，如果聊天内容不为空，则先在聊天消息与系统消息显示区域中显示该聊天内容，用户名为 SYS，event 参数值为 OUT。接下来调用 jWebSocketJSONClient 对象的 broadcastText 方法将该聊天内容广播给其他所有登录到 jWebSocket 聊天服务器的用户。如果广播失败，则在聊天消息与系统消息显示区域中显示相应的错误消息，用户名为 SYS，event 参数值为 OUT。同时将对对话文本框中内容清空。

#### ❑ btnLogout\_onclick 函数

用户单击退出按钮时调用 btnLogout\_onclick 函数。该函数首先调用 jWebSocketJSONClient 对象的 close 方法断开客户端与服务器端之间的连接，然后在聊天消息与系统消息显示区域中显示一条消息，文字为：“用户”+ 客户端登录到 jWebSocket 聊天服务器时所使用的用户名 + “退出聊天室：” + 断开客户端与服务器端之间连接操作的消息字符串。如果成功断开客户端与服务器端之间的连接，则将退出按钮与发送按钮设定为无效状态。

#### ❑ window\_onunload 函数

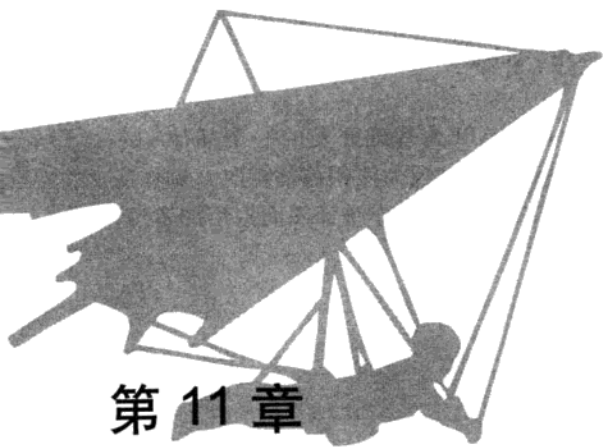
关闭页面时调用 window\_onunload 函数。该函数判断页面上的退出按钮是否为有效状态，如果为有效状态则表示客户端与 jWebSocket 服务器端之间已建立连接，这时调用 jWebSocketJSONClient 对象的 close 方法断开客户端与服务器端之间的连接。

## 10.8 本章小结

本章对 jWebSocket 通信框架中客户端与服务器端的一些基本概念，以及如何使用

jWebSocket 通信框架来开发 WebSocket 通信程序作了系统介绍，希望读者通过本章的阅读，对 jWebSocket 通信框架有一个基本的认识，能够运用 jWebSocket 通信框架开发出属于自己的 WebSocket 通信应用程序。

下一章将介绍以 HTML 5 中的 canvas 元素与 Canvas API 为基础开发的一个插件——RGraph 统计图制作插件。该插件被专门用来制作各种统计图，因为使用该插件时，制作统计图的工作被分散到客户端完成，所以可以大量节省服务器端因制作统计图而占用的资源，同时绘制出各种复杂的，功能比较强大的统计图。



## 第 11 章

# RGraph 统计图制作插件

### 本章内容

- ☐ 概述
- ☐ 绘制统计图时所用到的公共属性
- ☐ 绘制柱状图
- ☐ 绘制折线图
- ☐ 绘制饼图
- ☐ 绘制横向柱状图
- ☐ 绘制雷达图
- ☐ 增强用户体验
- ☐ 本章小结

本章介绍一个国外开发的利用 HTML 5 中 canvas 元素进行专业统计图制作的插件，该插件的名称为 RGraph，读者可以从其官方网站（<http://www.rgraph.net/>）上下载免费版本。使用该插件可以在利用 HTML 5 开发的网站或 Web 应用程序中制作出各种功能强大的统计图。本章对该插件进行概略介绍，重点介绍几个常见统计图的制作方法。

## 11.1 概述

### 11.1.1 HTML 5 版统计图插件的优越性

在 HTML 5 中，新增了一个非常重要的元素——canvas 元素。使用该元素，可以在页面中直接进行各种复杂图形的制作。使用该元素绘制统计图，比之前使用服务器端控件生成统计图的方法更加具有优越性，因为使用该元素后，绘制统计图的工作直接在客户端进行，而不是在服务器端完成。这不仅意味着不再占用服务器端资源，而且可以直接利用客户端计算机的强大资源，大大提高了绘制统计图的速度。控制 canvas 图形绘制的脚本代码是可以被压缩（例如，在使用 Apache 服务器的时候，mod\_gzip 将自动执行代码压缩工作）和被缓存的，大幅度减少了对带宽的占用。

设想一下，由于客户端的访问，服务器端每天需要创建 100 000 幅统计图，这对服务器端来说，无疑会占用非常巨大的资源。使用 RGraph 统计图制作插件可以将资源占用减少到接近零的程度，因为所有创建统计图的工作都在客户端完成，就像渲染 HTML 网页一样，服务器端只负责发送数据，不再负责统计图的生成与发送，同时带宽的占用情况也得到大大改善。

另外，由于统计图是依靠 JavaScript 生成的，因此在查看这个显示统计图的 HTML 网页的时候，该网页可以处于离线状态。

目前该插件受到了 Firebox 4、Google Chrome 10、Opera 11、Safari 5 和 IE 9 等浏览器的支持。

### 11.1.2 使用 RGraph 插件

RGraph 插件使用起来非常方便，具体分为以下几个步骤。

1) 在页面中引入 RGraph 插件所使用的脚本文件，其中 RGraph.common.core.js 脚本文件（插件的核心代码脚本文件）是必须引入的，代码如下所示。

```
<script src="RGraph.common.core.js"></script>
```

2) 根据页面中需要绘制的统计图的类型与功能引入对应的脚本文件，例如，绘制柱状图时需要引入 RGraph.bar.js 脚本文件，代码如下所示。

```
<script src="RGraph.bar.js"></script>
```



3) 添加 canvas 元素 (用来绘制与显示统计图), 代码类似如下所示。

```
<canvas id="myCanvas" width="600" height="300">
    [ 您的浏览器不支持 canvas 元素 ]
</canvas>
```

4) 创建统计图, 代码类似如下所示。

```
<script>
function window_onload()
{
    var data = [280, 45, 133, 166, 84, 259, 266, 960, 219, 311, 67, 89];
    var myGraph = new RGraph.Bar('myCanvas', data);
    myGraph.Set('chart.labels', ['1月','2月','3月','4月','5月','6月',
    '7月','8月','9月','10月','11月','12月']);
    myGraph.Set('chart.gutter',35);
    myGraph.Draw();
}
</script>
```

### 11.1.3 使用服务器端数据

要想利用服务器端特别是数据库中返回的数据来绘制统计图, 实现起来也非常容易, 只需让服务器端将数据发送到客户端即可。例如, 在 ASP.NET 中, 在服务器端生成一个数组 (该数组可由数据库中返回的数据组成), 将该数组发送到客户端即可, 代码类似如下所示。

```
namespace HTML5TEST
{
    public partial class Default : System.Web.UI.Page
    {
        public int[] dataArray;
        protected void Page_Load(object sender, EventArgs e)
        {
            dataArray=new int[]{280, 45, 133, 166, 84, 259, 266, 960, 219, 311,
            67, 89}; // 该数组可由数据库中返回的数据组成
            .....
        }
        .....
    }
}
```

在客户端的 JavaScript 脚本代码中, 可直接使用服务器端的这个 dataArray, 代码类似如下所示。

```
<script language="javascript">
function window_onload()
{
    // 将服务器端的数组传给客户端的数组
    var data=new Array(12);
    <%
```

```

string iniArr;
for(int i=0;i <12;i++)
{
    iniArr+="data["+i+"]="+dataArray[i]+" ";
}
%>
<%=iniArr%>
var myGraph = new RGraph.Bar('myCanvas', data);
myGraph.Set('chart.labels', ['1月','2月','3月','4月','5月','6月',
'7月','8月','9月','10月','11月','12月']);
myGraph.Set('chart.gutter',35);
myGraph.Draw();
}
</script>

```

在 PHP 中，可以采用如下所示的代码。

```

<?php
// 服务器端数组，可由数据库中返回的数据组成
$myData = join(',', array(280, 45, 133, 166, 84, 259, 266, 960, 219, 311,
67, 89));

// 生成客户端脚本代码
print('<script src="RGraph.common.core.js"></script>' . "\n");
print('<script src="RGraph.bar.js"></script>' . "\n\n");
print('<canvas id="myCanvas" width="600" height="300">[ 您的浏览器不支持
canvas 元素 ]</canvas>' . "\n\n");
print('<script>' . "\n");
print('var data = [' . $myData . '];' . "\n\n");
print('var myGraph = new RGraph.Bar("myCanvas", data);' . "\n");
print('myGraph.Set("chart.labels", ["1月","2月","3月","4月","5月",
"6月","7月","8月","9月","10月","11月","12月"]);' . "\n");
print('myGraph.Set("chart.gutter",35);' . "\n");
print('myGraph.Draw();');
print('</script>');
?>

```

## 11.2 绘制统计图时所用到的公共属性

接下来介绍一下使用 RGraph 插件绘制各种统计图时会使用到的一些公共属性。对统计图使用属性的方法如下所示。

```
myGraph.Set('name', 'value');//myGraph 代表统计图对象
```

### 1. 整体设置相关属性

#### □ chart.width

用来设置柱状图的整体宽度。例如，可以将该值设为 600，将 canvas 元素宽度设为 650，为柱状图左侧（或右侧）的坐标轴数字或文字留下 50px 的宽度。

不设置该属性值时柱状图宽度为显示柱状图所用的 canvas 元素的宽度。

#### □ chart.height

用来设置柱状图的整体高度。例如，可以将该值设为 250，将 canvas 元素宽度设为 300，为柱状图下侧或上侧的坐标轴数字或文字留下 50px 的高度。

不设置该属性值时柱状图高度为显示柱状图所用的 canvas 元素的高度。

#### □ chart.gutter

用于设置统计图标签所使用的空间尺寸，默认值为 25。如果标签文字由于空间不够而没有被全部显示出来，可以增加这个尺寸。

在柱状图或折线图等具有坐标轴的统计图中，统计图标签位于垂直坐标轴旁边或水平坐标轴下部。垂直坐标轴旁的标签文字显示垂直坐标轴的刻度数字（如 500、1000、1500、2000 等），水平坐标轴下部的标签文字显示水平坐标轴中的绘制单位（如 1 月、2 月、3 月等）。

在饼图或其他没有坐标轴的统计图中，统计图标签位于每个绘制区域（如饼图中的每个饼块）旁边，用于说明每个绘制区域的具体数值。

### 2. 颜色相关属性

chart.colors，通过数组来指定绘制统计图中每个绘制区域所使用的填充颜色（例如，柱状图中为每根柱子的颜色，饼图中为每个饼块所使用的颜色）。默认值为数组 ['rgb(0,0,255)', '#0f0', '#00f', '#ff0', '#0ff', '#0f0']。

### 3. 统计图标签相关属性

#### □ chart.text.color

用于设置统计图的标签文字的文字颜色。默认为黑色。

#### □ chart.text.size

用于设置统计图的标签文字的文字大小。默认为 10px。

#### □ chart.text.font

用于设置坐标轴数字或文字所使用的字体。默认值为 Verdana。

#### □ chart.labels

用一个数组来显示统计图的标签文字。在柱状图或折线图等具有坐标轴的统计图中，该属性只用于指定水平坐标轴下部的标签文字。默认值为一个空数组。

### 4. 统计图标题相关属性

#### □ chart.title

用于设置统计图的标题文字。默认值为 null（不设置标题）。

#### □ chart.title.background

用于设置统计图标题的背景色。默认值为 null（不设置背景色）。

#### □ chart.title.color

用于设置统计图标题的文字颜色。默认值为 black（黑色）。

**□ chart.title.hpos**

用于设置统计图标题的横向位置。该值必须为从 0 到 1 中的一个数字，然后将该数字乘以 canvas 元素的宽度，得到坐标轴标题文字中心点的 x 坐标。默认值为 null（中央显示）。

**□ chart.title.vpos**

用于设置统计图标题的纵向位置。该值必须为从 0 到 1 中的一个数字，然后用该数字乘以 chart.gutter 属性中所设置的值，得到统计图标题的纵向坐标。默认值为 null（自动设置）。

**5. 阴影相关属性****□ chart.shadow**

用于设置是否在统计图中使用阴影效果。默认值为 false（不绘制）。

**□ chart.shadow.color**

用于设置统计图中阴影的颜色。默认值为 #666。

**□ chart.shadow.offsetx**

用于设置统计图中阴影的横向宽度。默认值为 3。

**□ chart.shadow.offsety**

用于设置统计图中阴影的垂直方向的投射位移量。默认值为 3。

**□ chart.shadow.blur**

用于设置统计图中阴影的向外晕开程度。默认值为 3。

**6. 用户交互相关属性****□ chart.contextmenu**

使用一个数组来指定上下文菜单中所有菜单项。默认值为一个空数组。

**□ chart.resizable**

允许用户通过拖曳操作来改变统计图尺寸。默认值为 false。

**□ chart.tooltips**

一个带有 index 编号的数组，其中存放用户单击每根柱子时显示的工具条提示信息，该提示信息可以被书写成 HTML 的形式。默认值为 null。

**□ chart.tooltips.effect**

显示工具条提示信息时的动态效果。可以指定的值为 fade（淡入）或 expand（扩展）。默认值为 fade（淡入）。

**□ chart.tooltips.event**

用于指定触发工具条提示的事件。可以指定的值为 onclick 与 onmousemove。默认值为 onclick。

**□ chart.tooltips.css.class**

工具条提示信息所使用的 css 类名。默认值为 Rgraph\_tooltip。

**□ chart.tooltips.override**

如果用户自己处理工具条提示信息的显示方式，可以在该值中指定一个自定义函数。默

认值为 null。

❑ `chart.annotatable`

用于设置是否能对统计图使用注解（例如，用户可以在统计图上自己绘制注解）。默认值为 false。

❑ `chart.annotate.color`

用于设置绘制注解时所使用的颜色。默认值为 black（黑色）。

## 7. 图例相关属性

❑ `chart.key`

用于设置一个存放图例信息的数组。默认值为一个空数组。

❑ `chart.key.background`

用于设置柱状图所用图例的背景色。默认为白色，可以设置为其他颜色，或设置为类似 rgba（255,255,255,0.7）的值来使其产生透明效果。默认值为白色。

❑ `chart.key.halign`

不通过指定坐标点的方式来设置其位置，而是简单指定图例为左对齐方式或右对齐方式。默认值为 right。

❑ `chart.key.position`

用于设置图例是否被绘制在一个单独的图例区域中。默认值为 graph（在图例区域中绘制图例），可以被设置为 gutter（直接在坐标轴旁边绘制图例文字）。

❑ `chart.key.position.x`

用于设置指定图例在 X 轴上的坐标点。

❑ `chart.key.position.y`

用于设置指定图例在 Y 轴上的坐标点。

❑ `chart.key.position.gutter.boxed`

如果在图例区域中绘制图例，该值用来指定是否对图例区域使用背景色。默认值为 true。

❑ `chart.key.shadow`

用于设置是否对图例绘制阴影。默认值为 false。

❑ `chart.key.shadow.color`

用于设置图例阴影的颜色。默认值为 #666。

❑ `chart.key.shadow.blur`

用于设置图例阴影的晕开半径。默认值为 3。

❑ `chart.key.shadow.offsetx`

用于设置图例阴影在 X 轴方向上的位移值。

❑ `chart.key.shadow.offsety`

用于设置图例阴影在 Y 轴方向上的位移值。

#### ❑ chart.key.rounded

用于设置是否将图例区域的边框绘制成圆角边框。如果直接在柱状图旁绘制图例文字，那么该属性不起作用。默认值为 false。

#### ❑ chart.key.color.shape

用于设置图例中的颜色块的形状。可设置为 square（方形）、circle（圆形）和 line（直线）。默认值为 square（方形）。

#### ❑ chart.key.linewidth

用于设置图例边框的线宽。默认值为 1。

### 8. 统计图放大相关属性

#### ❑ chart.zoom.mode

用于指定使用缩略图模式还是 canvas 元素模式放大显示统计图中某个局部。可以指定的值为 thumbnail（缩略图模式）与 canvas（canvas 元素模式）。默认值为 canvas。

#### ❑ chart.zoom.factor

用于设置放大统计图时所使用的放大倍数。默认值为 1.5 倍。

#### ❑ chart.zoom.fade.in

用于设置是否使用淡入的方式来放大显示统计图。默认值为 true。

#### ❑ chart.zoom.fade.out

用于设置是否使用淡出的方式来结束统计图的放大显示。默认值为 true。

#### ❑ chart.zoom.delay

用于设置缩放时帧与帧之间的间隔毫秒数。默认值为 50。

#### ❑ chart.zoom.frames

用于设置执行缩放动画时所使用的帧数。默认值为 10。

#### ❑ chart.zoom.shadow

用于设置被放大显示的统计图是否具有阴影效果。

#### ❑ chart.zoom.thumbnail.width

用于设置当缩放模式为缩略图模式时的缩略图宽度。默认值为 75。

#### ❑ chart.zoom.thumbnail.height

用于设置当缩放模式为缩略图模式时的缩略图高度。默认值为 75。

#### ❑ chart.zoom.background

用于设置统计图的放大图是否具有一个深色半透明的背景（可以看见网页）。默认值为 true。

### 9. 格式相关属性

#### ❑ chart.units.pre

用于设置在数值类型的标签文字中增加一个单位。该单位被放置在数值前边，例如可以将数值指定为“\$50”。默认值为 none（不指定）。

❑ `chart.units.post`

用于设置在数值类型的标签文字中增加一个单位。该单位被放置在数值后边，例如可以将数值指定为“50ms”。默认值为 `none`（不指定）。

## 10. 其他属性

`chart.align`，用于指定统计图的横向对齐方式，可以指定的值为 `left`（左对齐）、`center`（中央对齐）与 `right`（右对齐）。默认值为 `center`。

## 11.3 绘制柱状图

本节将具体介绍如何使用 RGraph 插件绘制柱状图。在讲解示例前，先来看一下在绘制柱状图时所使用的属性。

### 11.3.1 绘制柱状图时所用到的属性

除了前面讲到的绘制统计图时会使用到的公共属性外，绘制柱状图时将会使用的属性如下。

#### 1. 背景相关属性

❑ `chart.background.barcolor1`

用于设置柱状图背景网格中的奇数行（从上往下数）的背景颜色（默认为白色）。

❑ `chart.background.barcolor2`

用于设置柱状图背景网格中的偶数行（从上往下数）的背景颜色（默认为白色）。

❑ `chart.background.grid`

用于指定是否显示背景网格。默认值为 `true`（显示）。

❑ `chart.background.grid.color`

用于设置背景网格线的颜色。默认颜色为 `#ddd`。

❑ `chart.background.grid.hsize`

用于设置背景网格的每格宽度。默认值为 40。

❑ `chart.background.grid.vsize`

用于设置背景网格的每格高度。默认值为 18。

❑ `chart.background.grid.width`

用于设置背景网格线宽。允许属性值不为整数（如 0.5）。默认值为 1。

❑ `chart.background.grid.border`

用于设置是否在背景网格周围绘制一个边框线。默认值为 `true`。

❑ `chart.background.grid.hlines`

用于设置是否绘制背景网格中的水平线。默认值为 `true`。

❑ `chart.background.grid.vlines`

用于设置是否绘制背景网格中的垂直线。默认值为 `true`。

#### ☐ chart.background.grid.autofit

不指定背景网格中每格宽度或高度，只指定绘制多少条垂直线与水平线，然后采用自动计算方式。默认值为 false（不采用这种方式）。

#### ☐ chart.background.grid.autofit.numhlines

用于指定当使用自动计算方式绘制背景网格时需要绘制的水平线条数。默认值为 7。

#### ☐ chart.background.grid.autofit.numvlines

用于指定当使用自动计算方式绘制背景网格时需要绘制的垂直线条数。默认值为 20。

#### ☐ chart.background.grid.autofit.align

用于指定背景网格线是否自动与坐标轴刻度对齐。默认值为 false。

### 2. 坐标轴相关属性

#### ☐ chart.noaxes

用于指定是否绘制坐标轴。默认值为 false（绘制坐标轴）。

#### ☐ chart.noaxis

用于指定是否不绘制水平坐标轴。默认值为 false（绘制水平坐标轴）。

#### ☐ chart.noyaxis

用于指定是否绘制垂直坐标轴。默认值为 false（绘制垂直坐标轴）。

### 3. 颜色相关属性

#### ☐ chart.strokecolor

用于设置绘制柱状图中每根柱子的边框颜色。默认值为 #666。

#### ☐ chart.colors.sequential

用于指定是否只能由相同的颜色来绘制柱状图中每根柱子的颜色。当该值为 false 时只能用指定的第一根柱子的颜色来绘制其他柱子的颜色。当该值为 true 时可以对每根柱子指定不同的颜色。默认值为 false。

### 4. 边距相关属性

chart.hmargin, 用于设置每根柱子与两侧网格线的相隔距离（以像素为单位）。默认值为 5。

### 5. 标签相关属性

#### ☐ chart.text.angle

用于设置柱状图水平坐标轴下部标签文字的显示角度。允许设置该属性值为：0、45 与 90。默认值为 0（横向显示）。

#### ☐ chart.labels.above

该值为 true 时，将会在每根柱子的上部显示一个具体说明该柱子所代表数值的数字。默认值为 false。

#### ☐ chart.labels.above.decimals

用于设置坐标轴每根柱子上部数值精度（允许精确到小数点后几位）。默认值为 0。



#### ❑ chart.labels.above.size

用于设置坐标轴每根柱子上部数值所使用的文字大小。默认值为 false（不指定）。

#### ❑ chart.labels.above.angle

用于设置坐标轴每根柱子上部数值的显示角度。允许的指定值范围为 -90 到 90。默认值为 null（横向显示）。

#### ❑ chart.labels.ingraph

用来指定一个被显示在每根柱子内部的文字数组。默认值为 null（不指定）。

#### ❑ chart.ylabels

用于设置是否绘制垂直坐标轴（即纵向坐标轴或 Y 坐标轴）。默认值为 true。

#### ❑ chart.ylabels.count

用于设置垂直坐标轴上显示多少个单位（或者说显示多少个垂直坐标轴文字），可以为任何数字，但垂直坐标轴上的最大数字除以这个数字后必须得到一个整数，否则所有垂直坐标轴文字均不显示。默认值为 5（纵向显示 5 个单位）。

#### ❑ chart.xlabels.offset

用于设置坐标轴底部文字离开水平坐标轴的位移距离。默认值为 0。

#### ❑ chart.numyticks

用于设置垂直坐标轴上显示多少根刻度线。默认值为 10。

### 6. 标题相关属性

#### ❑ chart.title.xaxis

用于设置水平坐标轴的标题文字。默认值为 none（不设置标题文字）。

#### ❑ chart.title.yaxis

用于设置垂直坐标轴的标题文字。默认值为 none（不设置标题文字）。

#### ❑ chart.title.xaxis.pos

用该值乘以 chart.gutter 属性中所设置的值得到水平坐标轴标题文字的位置。默认值为 0.25。

#### ❑ chart.title.yaxis.pos

用该值乘以 chart.gutter 属性中所设置的值得到垂直坐标轴标题文字的位置。默认值为 0.25。

### 7. 格式相关属性

chart.ymax，用于设置 Y 坐标轴上数字的最大值。如果不指定则会被自动计算出来。默认值为 null（自动计算）。

### 8. 用户交互相关属性

#### ❑ chart.crosshairs

该值被设为 true 时，当前鼠标位置处会显示一个十字光标。默认值为 false。

#### ❑ chart.crosshairs.linewidth

用于设置十字光标的线宽。默认值为 1。

☐ chart.crosshairs.color

用于设置十字光标的颜色。默认值为 #333。

☐ chart.adjustable

用于设置是否允许用户调整统计图。默认值为 false。

### 9. 其他属性

☐ chart.line

以一个折线图作为属性值，用于将柱状图与折线图结合成为一张统计图。

☐ chart.xaxispos

用于指定水平坐标轴的位置，可设定的值为 bottom（底部）或 center（中央）。默认值为 bottom。

☐ chart.axis.color

用于指定坐标轴的颜色。默认值为 black（黑色）。

☐ chart.grouping

在绘制分组柱状图时指定分组形式。可以设定的值为 grouped（在 X 轴的同一个单位上绘制多根柱子）或 stacked（将一根柱子分为几层颜色）。默认值为 grouped。

## 11.3.2 示例程序

接下来看一个使用 RGraph 插件制作柱状图的示例程序，该示例程序的功能为显示 2010 年常州第一百货公司长虹彩电销售情况的统计柱状图。该示例程序在浏览器中打开时的显示效果如图 11-1 所示。

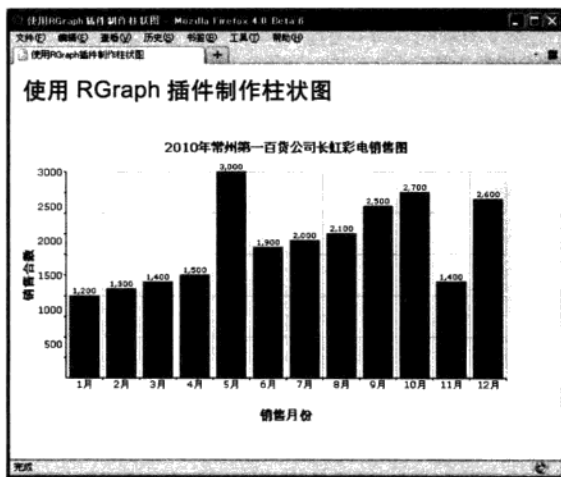


图 11-1 使用 RGraph 插件制作柱状图

该示例程序的完整代码如代码清单 11-1 所示。

代码清单 11-1 使用 RGraph 插件制作柱状图

---

```

<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title> 使用 RGraph 插件制作柱状图 </title>
<script src="RGraph.common.core.js"></script>
<script src="RGraph.bar.js"></script>
<script>
function window_onload()
{
    // 绘制柱状图, 指定数据
    myGraph = new RGraph.Bar('myCanvas', [1200,1300,1400,1500,3000,1900,2000,
    2100,2500,2700,1400,2600]);
    // 指定统计图标题
    myGraph.Set('chart.title', '2010 年常州第一百货公司长虹彩电销售图 ');
    // 指定 X 轴标题
    myGraph.Set('chart.title.xaxis', '销售月份 ');
    // 指定 Y 轴标题
    myGraph.Set('chart.title.yaxis', '销售台数 ');
    // 指定 X 轴的坐标轴文字
    myGraph.Set('chart.labels', ['1 月 ', '2 月 ', '3 月 ', '4 月 ', '5 月 ', '6 月 ', '7 月 ',
    '8 月 ', '9 月 ', '10 月 ', '11 月 ', '12 月 ']);
    // 指定 Y 轴的坐标轴文字
    myGraph.Set('chart.ylabels.specific', ['3000', '2500', '2000', '1500',
    '1000', '500']);
    // 指定在坐标轴顶部绘制说明销售数量的文字
    myGraph.Set('chart.labels.above', true);
    // 指定网格自动与坐标轴单位对齐
    myGraph.Set('chart.background.grid.autofit', true);
    myGraph.Set('chart.background.grid.autofit.align', true);
    // 指定标签文字所使用的空间尺寸
    myGraph.Set('chart.gutter', 65);
    // 绘制柱状图
    myGraph.Draw();
}
</script>
</head>
<body onload="window_onload()">
<h1> 使用 RGraph 插件制作柱状图 </h1>
<canvas id="myCanvas" width="700" height="400">
    [ 您的浏览器不支持 canvas 元素 ]
</canvas>
</body>
</html>

```

---

### 11.3.3 使用 obj.getBar 方法

在绘制柱状图时, 可以使用一个 obj.getBar 方法来使开发者或用户知道哪个柱子被单

击, 或获得鼠标焦点。该方法返回一个数组, 其中存有以下信息:

- 1) 用于绘制统计图对象的 canvas 元素。
- 2) 被单击柱子的绘制起点在 X 轴上的坐标点。
- 3) 被单击柱子的绘制起点在 Y 轴上的坐标点。
- 4) 被单击柱子的宽度。
- 5) 被单击柱子的高度。
- 6) 被单击柱子的序号 (标示第几根柱子被单击)。

下面看一个使用该方法的示例。该示例在代码清单 11-1 的基础上稍加修改, 每次单击柱状图中的一根柱子都会使这根柱子变为不同的颜色。代码如代码清单 11-2 所示。

代码清单 11-2 每次单击柱子将使该柱子变为不同的颜色

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title> 使用 RGraph 插件制作柱状图 </title>
<script src="RGraph.common.core.js"></script>
<script src="RGraph.bar.js"></script>
<script>
// 指定颜色数组, 用于变换颜色
var color=new Array;
color[0]='red';
color[1]='green';
color[2]='blue';
function window_onload()
{
    // 绘制柱状图, 指定数据
    myGraph = new RGraph.Bar('myCanvas', [1200,1300,1400,1500,3000,1900,2000,
    2100,2500,2700,1400,2600]);
    // 指定统计图标题
    myGraph.Set('chart.title', '2010 年常州第一百货公司长虹彩电销售图 ');
    // 指定 X 轴标题
    myGraph.Set('chart.title.xaxis', '销售月份 ');
    // 指定 Y 轴标题
    myGraph.Set('chart.title.yaxis', '销售台数 ');
    // 指定 X 轴的坐标轴文字
    myGraph.Set('chart.labels', ['1 月 ', '2 月 ', '3 月 ', '4 月 ', '5 月 ', '6 月 ', '7 月 ',
    '8 月 ', '9 月 ', '10 月 ', '11 月 ', '12 月 ']);
    // 指定 Y 轴的坐标轴文字
    myGraph.Set('chart.ylabels.specific', ['3000', '2500', '2000', '1500',
    '1000', '500']);
    // 指定在坐标轴顶部绘制说明销售数量的文字
    myGraph.Set('chart.labels.above', true);
    // 指定网格自动与坐标轴单位对齐
    myGraph.Set('chart.background.grid.autofit', true);
    myGraph.Set('chart.background.grid.autofit.align', true);
    // 指定标签文字所使用的空间尺寸
```

```

myGraph.Set('chart.gutter',65);
// 绘制柱状图
myGraph.Draw();
// 注册控件
RGraph.Register(myGraph);
var i=0;// 填充时使用颜色的颜色序号
myGraph.canvas.onclick = function (e)
{
    RGraph.Redraw();// 重绘统计图
    var canvas = e.target;// 获取被单击的 canvas 元素
    var context = canvas.getContext('2d');// 获取该 canvas 元素的图形上下文对象
    var obj = canvas.__object__;// 获取统计图对象
    var coords = obj.getBar(e);// 获取被单击的柱子信息

    if (coords) {
        var top = coords[1];// 获取被单击柱子在 X 轴上的坐标起点
        var left = coords[2];// 获取被单击柱子在 Y 轴上的坐标起点
        var width = coords[3];// 获取被单击柱子的宽度
        var height = coords[4];// 获取被单击柱子的高度

        context.beginPath();// 开始创建路径
        context.strokeStyle = 'black';// 指定柱子的边框颜色
        context.fillStyle =color[i%3];// 指定柱子的填充颜色
        i+=1;// 指定下次使用颜色的颜色编号
        context.strokeRect(top, left, width, height);// 绘制柱子边框
        context.fillRect(top, left, width, height);// 填充柱子
    }
}
}
</script>
</head>
<body onload="window.onload()">
<h1> 使用 RGraph 插件制作柱状图 </h1>
<canvas id="myCanvas" width="700" height="400">
    [ 您的浏览器不支持 canvas 元素 ]
</canvas>
</body>
</html>

```

### 11.3.4 绘制分组柱状图

接下来看两个绘制分组柱状图的示例，第一个示例为在水平坐标轴的一个坐标单位上绘制多根柱子，第二个示例为在一根柱子上使用多种颜色。

#### 1. 在一个坐标单位上绘制多根柱子

首先来看如何在水平坐标轴的一个坐标单位上绘制多根柱子，本示例为 2010 年常州第一百货公司彩电销售情况的统计柱状图，其中只统计了长虹彩电与康佳彩电的销售数量。在

一个月份中绘制两根柱子，一根柱子为长虹彩电在该月份的销售数量，另一根柱子为康佳彩电在该月份的销售数量。分别使用蓝色与绿色绘制两种不同的柱子，在图例中说明蓝色柱子表示长虹彩电的销售数量，绿色柱子表示康佳彩电的销售数量，在每根柱子顶部绘制该柱子所代表的销售数量。

该示例在浏览器中的显示效果如图 11-2 所示。

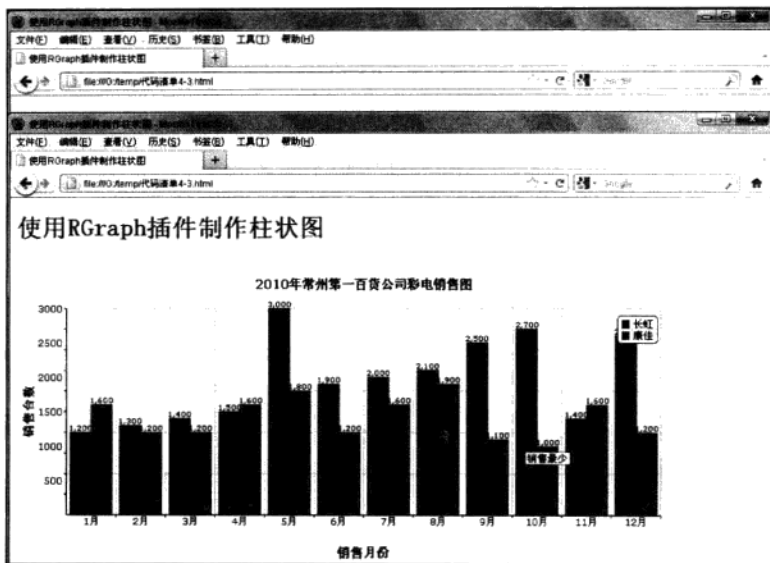


图 11-2 在一个坐标单位上绘制多根柱子

该示例的代码如代码清单 11-3 所示，在代码清单中具体标注了使用哪些属性，以及这些属性的用途。

代码清单 11-3 在一个坐标单位上绘制多根柱子

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title> 使用 RGraph 插件制作柱状图 </title>
<script src="RGraph.common.core.js"></script>
<script src="RGraph.bar.js"></script>
<script>
function window_onload()
{
    // 绘制分组柱状图，指定数据
    myGraph = new RGraph.Bar('myCanvas', [[1200,1600], [1300,1200], [1400,1200],
        [1500,1600], [3000,1800], [1900,1200], [2000,1600], [2100,1900],
        [2500,1100], [2700,1000], [1400,1600], [2600,1200]]);
    // 指定统计图标题
    myGraph.Set('chart.title', '2010 年常州第一百货公司彩电销售图');
```

```

// 指定 X 轴标题
myGraph.Set('chart.title.xaxis', '销售月份');
// 指定 Y 轴标题
myGraph.Set('chart.title.yaxis', '销售台数');
// 指定柱状图图例被绘制在图例区域中
myGraph.Set('chart.key.position', 'graph');
// 指定图例文字
myGraph.Set('chart.key', ['长虹', '康佳']);
// 指定柱子颜色
myGraph.Set('chart.colors', ['blue', 'green']);
// 指定 X 轴的坐标轴文字
myGraph.Set('chart.labels', ['1月', '2月', '3月', '4月', '5月', '6月', '7月',
'8月', '9月', '10月', '11月', '12月']);
// 指定 Y 轴的坐标轴文字
myGraph.Set('chart.ylabels.specific', ['3000', '2500', '2000', '1500',
'1000', '500']);
// 指定在坐标轴顶部绘制的说明销售台数的文字
myGraph.Set('chart.labels.above', true);
// 用文字说明销售量最少的柱子
myGraph.Set('chart.labels.ingraph', [19, '销售最少']);
// 指定网格自动与坐标轴单位对齐
myGraph.Set('chart.background.grid.autofit', true);
myGraph.Set('chart.background.grid.autofit.align', true);
// 指定标签文字所使用的空间尺寸
myGraph.Set('chart.gutter', 65);
myGraph.Draw();
}
</script>
</head>
<body onload="window.onload()">
<h1>使用 RGraph 插件制作柱状图</h1>
<canvas id="myCanvas" width="900" height="400">
    [您的浏览器不支持 canvas 元素]
</canvas>
</body>
</html>

```

## 2. 将一根柱子分为几层颜色

接下来介绍如何使用多种颜色绘制柱状图中的每一根柱子。本示例的功能与“在一个坐标单位上绘制多根柱子”示例的功能大致相同，都是显示 2010 年常州第一百货公司长虹与康佳这两种彩电的销售情况，但是本示例的说明方法不是在一个月份中使用两种颜色的柱子，而是将长虹与康佳这两种彩电每月的销量绘制在同一根柱子中。这里使用两种不同的颜色进行绘制，在下部绘制用来说明康佳彩电销售数量的绿色柱子，在上部绘制用来说明长虹彩电销售数量的蓝色柱子。使用这种方法的好处在于既可以看出两种彩电各自的销售数量，又可以直接看出这两种彩电的销售总数量。

该示例在浏览器中的显示效果如图 11-3 所示。

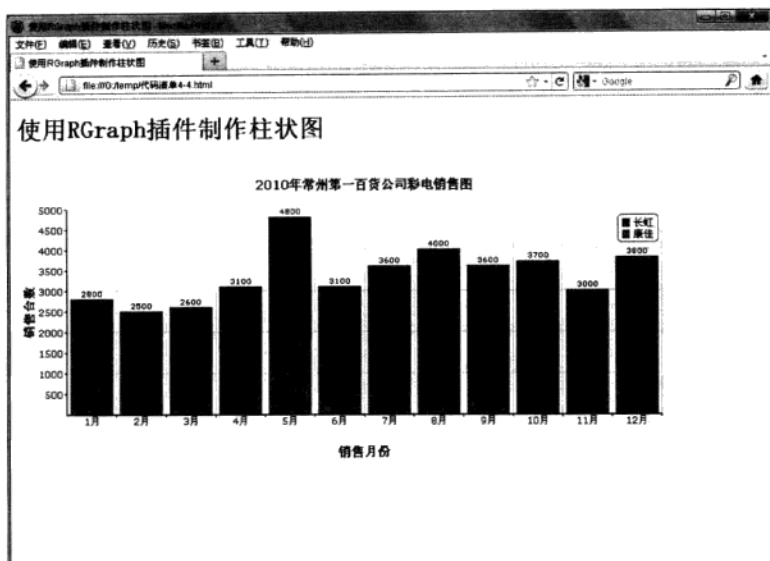


图 11-3 在一根柱子中使用两种颜色

这个示例的程序代码也与代码清单 11-3 的代码大致相同，只是将 `chart.grouping` 属性的属性值设成了 `stacked`，并且将在 `chart.ylabels.specific` 中指定的垂直坐标轴上的坐标数字增加到了 5000（原来为单种彩电的最大销售数量，现在修改为两种彩电总的最大销售数量），完整代码如代码清单 11-4 所示。在代码清单中具体标注了使用哪些属性以及这些属性的作用。

代码清单 11-4 在一根柱子中使用两种颜色

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title> 使用 RGraph 插件制作柱状图 </title>
<script src="RGraph.common.core.js"></script>
<script src="RGraph.bar.js"></script>
<script>
function window_onload()
{
    // 绘制分组柱状图，指定数据
    myGraph = new RGraph.Bar('myCanvas', [[1200,1600],[1300,1200],[1400,1200],
        [1500,1600],[3000,1800],[1900,1200],[2000,1600],[2100,1900],
        [2500,1100],[2700,1000],[1400,1600],[2600,1200]]);
    // 指定统计图标题
    myGraph.Set('chart.title','2010 年常州第一百货公司彩电销售图');
    // 指定 X 轴标题
    myGraph.Set('chart.title.xaxis','销售月份');
    // 指定 Y 轴标题
```



```

myGraph.Set('chart.title.yaxis','销售台数');
// 指定柱状图图例被绘制在图例区域中
myGraph.Set('chart.key.position', 'graph');
// 指定图例文字
myGraph.Set('chart.key', ['长虹', '康佳']);
// 指定柱子颜色
myGraph.Set('chart.colors', ['blue', 'green']);
// 指定 X 轴的坐标轴文字
myGraph.Set('chart.labels', ['1月', '2月', '3月', '4月', '5月', '6月', '7月',
'8月', '9月', '10月', '11月', '12月']);
// 指定 Y 轴的坐标轴文字
myGraph.Set('chart.ylabels.specific', ['5000', '4500', '4000', '3500',
'3000', '2500', '2000', '1500', '1000', '500']);
// 指定在坐标轴顶部绘制的说明销售总数量的文字
myGraph.Set('chart.labels.above', true);
// 指定网格自动与坐标轴单位对齐
myGraph.Set('chart.background.grid.autofit', true);
myGraph.Set('chart.background.grid.autofit.align', true);
// 指定标签文字所使用的空间尺寸
myGraph.Set('chart.gutter', 65);
// 设置分组柱状图的绘制方式
myGraph.Set('chart.grouping', 'stacked');
myGraph.Draw();
}
</script>
</head>
<body onload="window.onload()">
<h1>使用 RGraph 插件制作柱状图 </h1>
<canvas id="myCanvas" width="900" height="400">
    [您的浏览器不支持 canvas 元素]
</canvas>
</body>
</html>

```

### 11.3.5 使用上下文菜单

最后来看一下在统计图中使用上下文菜单的方法。使用上下文菜单的步骤分为如下两步：

1) 添加对上下文菜单脚本文件的引用，代码如下所示。

```
<script src="RGraph.common.context.js"></script>
```

2) 添加使用统计图的 chart.contextmenu 属性，并且在该属性值中指定单击某个菜单项时所执行的函数，代码类似如下所示。

```

myGraph.Set('chart.contextmenu',
[['在一个坐标单位上绘制多根柱子', drawGroupedBar1], ['将一根柱子分为几层颜色',
drawGroupedBar2]]);

```

接下来介绍一个使用上下文菜单的示例。该示例的代码在代码清单 11-3 与代码清单

11-4 的基础上稍加修改,使得在同一程序中可以同时使用前面介绍的两种方法来绘制分组柱状图。画面打开时使用在一个坐标单位上绘制多根柱子的方法来绘制分组柱状图,单击上下文菜单中的“将一根柱子分为几层颜色”菜单项时使用一个坐标单位上只绘制一根柱子,将每根柱子分为几层颜色的方法来绘制分组柱状图,单击上下文菜单中的“在一个坐标单位上绘制多根柱子”菜单项时仍然使用在一个坐标单位上绘制多根柱子的方法来绘制分组柱状图。该示例的完整代码如代码清单 11-5 所示。

代码清单 11-5 使用上下文菜单

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title> 使用RGraph插件制作柱状图 </title>
<script src="RGraph.common.core.js"></script>
<script src="RGraph.bar.js"></script>
<script src="RGraph.common.context.js"></script>

<script>
function drawGroupedBar1()
{
    // 清除已绘制的柱状图
    canvas=document.getElementById("myCanvas");
    context=canvas.getContext('2d');
    context.clearRect(0,0,canvas.width,canvas.height);

    myGraph = new RGraph.Bar('myCanvas', [[1200,1600], [1300,1200], [1400,1200],
        [1500,1600], [3000,1800], [1900,1200], [2000,1600], [2100,1900],
        [2500,1100], [2700,1000], [1400,1600], [2600,1200]]);
    // 指定统计图标题
    myGraph.Set('chart.title', '2010 年常州第一百货公司彩电销售图');
    // 指定 X 轴标题
    myGraph.Set('chart.title.xaxis', '销售月份');
    // 指定 Y 轴标题
    myGraph.Set('chart.title.yaxis', '销售台数');
    // 指定柱状图图例被绘制在图例区域中
    myGraph.Set('chart.key.position', 'graph');
    // 指定图例文字
    myGraph.Set('chart.key', ['长虹', '康佳']);
    // 指定柱子颜色
    myGraph.Set('chart.colors', ['blue', 'green']);
    // 指定 X 轴的坐标轴文字
    myGraph.Set('chart.labels', ['1 月', '2 月', '3 月', '4 月', '5 月', '6 月', '7 月',
        '8 月', '9 月', '10 月', '11 月', '12 月']);
    // 指定 Y 轴的坐标轴文字
    myGraph.Set('chart.ylabels.specific', ['3000', '2500', '2000', '1500',
        '1000', '500']);
    // 指定在坐标轴顶部绘制说明销售台数的文字
    myGraph.Set('chart.labels.above', true);
}
```

```

// 用文字说明销售量最少的柱子
myGraph.Set('chart.labels.ingraph', [19, '销售最少']);
// 指定网格自动与坐标轴单位对齐
myGraph.Set('chart.background.grid.autofit', true);
myGraph.Set('chart.background.grid.autofit.align', true);
// 指定标签文字所使用的空间尺寸
myGraph.Set('chart.gutter', 65);
myGraph.Set('chart.contextmenu', [[' 在一个坐标单位上绘制多根柱子 ',
drawGroupedBar1],
[' 将一根柱子分为几层颜色 ', drawGroupedBar2]]);
myGraph.Draw();
}

function drawGroupedBar2()
{
    // 清除已绘制的柱状图
    canvas=document.getElementById("myCanvas");
    context=canvas.getContext('2d');
    context.clearRect(0,0,canvas.width,canvas.height);
    // 绘制分组柱状图, 指定数据
    myGraph = new RGraph.Bar('myCanvas', [[1200,1600], [1300,1200], [1400,1200],
    [1500,1600], [3000,1800], [1900,1200], [2000,1600], [2100,1900],
    [2500,1100], [2700,1000], [1400,1600], [2600,1200]]);
    // 指定统计图标题
    myGraph.Set('chart.title', '2010 年常州第一百货公司彩电销售图');
    // 指定 X 轴标题
    myGraph.Set('chart.title.xaxis', '销售月份');
    // 指定 Y 轴标题
    myGraph.Set('chart.title.yaxis', '销售台数');
    // 指定柱状图图例被绘制在图例区域中
    myGraph.Set('chart.key.position', 'graph');
    // 指定图例文字
    myGraph.Set('chart.key', ['长虹', '康佳']);
    // 指定柱子颜色
    myGraph.Set('chart.colors', ['blue', 'green']);
    // 指定 X 轴的坐标轴文字
    myGraph.Set('chart.labels', ['1 月 ', '2 月 ', '3 月 ', '4 月 ', '5 月 ', '6 月 ', '7 月 ',
    '8 月 ', '9 月 ', '10 月 ', '11 月 ', '12 月 ']);
    // 指定 Y 轴的坐标轴文字
    myGraph.Set('chart.ylabels.specific', ['5000', '4500', '4000', '3500',
    '3000', '2500', '2000', '1500', '1000', '500']);
    // 指定在坐标轴顶部绘制的说明销售总数量的文字
    myGraph.Set('chart.labels.above', true);
    // 指定网格自动与坐标轴单位对齐
    myGraph.Set('chart.background.grid.autofit', true);
    myGraph.Set('chart.background.grid.autofit.align', true);
    // 指定坐标轴文字的绘制空间
    myGraph.Set('chart.gutter', 65);
    myGraph.Set('chart.grouping', 'stacked');
    myGraph.Set('chart.contextmenu', [[' 在一个坐标单位上绘制多根柱子 ',

```

```

        drawGroupedBar1], ['将一根柱子分为几层颜色', drawGroupedBar2]]);
        myGraph.Draw();
    }

</script>
</head>
<body onload="drawGroupedBar1()">
<h1> 使用 RGraph 插件制作柱状图 </h1>
<canvas id="myCanvas" width="900" height="400">
    [您的浏览器不支持 canvas 元素]
</canvas>
</body>
</html>

```

---

## 11.4 绘制折线图

本节介绍如何使用 RGraph 插件绘制折线图。在讲解示例前，先来介绍绘制折线图时所使用到的属性。

### 11.4.1 绘制折线图时所用到的属性

除了前面讲到的绘制所有统计图都会使用到的属性外，绘制折线图时还会使用如下属性。

#### 1. 背景相关属性

☐ chart.background.barcolor1

设置折线图背景网格中的奇数行（从上往下数）的背景颜色（默认为白色）。

☐ chart.background.barcolor2

设置折线图背景网格中的偶数行（从上往下数）的背景颜色（默认为白色）。

☐ chart.background.grid

指定是否显示背景网格，默认值为 true（显示）。

☐ chart.background.grid.color

设置背景网格线的颜色，默认颜色为 #eee。

☐ chart.background.grid.hsize

设置背景网格的每格宽度。默认值为 25。

☐ chart.background.grid.vsize

设置背景网格的每格高度。默认值为 25。

☐ chart.background.grid.width

设置背景网格线宽。默认值为 1。

❑ `chart.background.grid.border`

设置是否在背景网格周围绘制一个边框线。默认值为 `true`。

❑ `chart.background.grid.hlines`

设置是否绘制背景网格中的水平线。默认值为 `true`。

❑ `chart.background.grid.vlines`

设置是否绘制背景网格中的垂直线。默认值为 `true`。

❑ `chart.background.grid.autofit`

不指定背景网格中每格宽度或高度，只指定绘制多少条垂直线与水平线，然后采用自动计算方式。默认值为 `false`（不采用这种方式）。

❑ `chart.background.grid.autofit.numhlines`

指定当使用自动计算方式绘制背景网格时需要绘制的水平线条数。默认值为 7。

❑ `chart.background.grid.autofit.numvlines`

指定当使用自动计算方式绘制背景网格时需要绘制的垂直线条数。默认值为 20。

❑ `chart.background.grid.autofit.align`

指定背景网格线是否自动与坐标轴刻度对齐。默认值为 `false`。

❑ `chart.backdrop`

指定是否在折线周围绘制一圈阴影。默认值为 `false`。

❑ `chart.backdrop.size`

指定折线阴影的宽度。默认值为 30。

❑ `chart.backdrop.alpha`

指定折线阴影的透明度。可设置的值范围为从 0（完全透明）到 1（不透明）。默认值为 0.2。

## 2. 标签相关属性

❑ `chart.labels.above`

该值为 `true` 时，将会在每段折线的起点的上部显示一个具体说明该起点处所代表数值的数字。默认值为 `false`。

❑ `chart.labels.above.size`

用于设置每段折线的起点的上部所绘制数值的文字大小。默认值为 8。

❑ `chart.labels.ingraph`

用来指定一个被显示在每段折线区域内部的文字数组。默认值为 `null`（不指定）。

❑ `chart.ylabels`

用于设置是否绘制垂直坐标轴（又称做纵向坐标轴或 Y 坐标轴）。默认值为 `true`。

❑ `chart.ylabels.invert`

用于设置是否将垂直坐标轴翻转绘制（即 0 在最顶部，越往下数值越大）。默认值为 `false`。

❑ `chart.ylabels.count`

用于设置垂直坐标轴上显示多少个单位（或者说显示多少个垂直坐标轴文字），可指定的值为 1、3、5 或者 10。默认值为 5（纵向显示 5 个单位）。

☐ chart.ylabels.inside

用于设置垂直坐标轴上的标签文字是否被绘制在统计图内侧。默认值为 false。

☐ chart.ylabels.inside.color

用于设置当垂直坐标轴上的标签文字被绘制在统计图内侧时，标签所使用的背景色。默认值为 rgba(255,255,255,0.5)（白色半透明）。

☐ chart.xlabels.inside

用于设置水平坐标轴上的标签文字是否被绘制在坐标轴上部。默认值为 false。

☐ chart.xlabels.inside.color

用于设置当水平坐标轴上的标签文字被绘制在坐标轴上部时，标签所使用的背景色。默认值为 rgba(255,255,255,0.5)（白色半透明）。

☐ chart.text.size

用于设置统计图的标签文字的文字大小。默认为 10px。

☐ chart.text.angle

用于设置统计图水平坐标轴下部标签文字的显示角度。默认值为 0（横向显示）

### 3. 边距相关属性

chart.hmargin, 用于设置折线起点与终点的外边距（离开背景网格线的距离）。默认值为 0。

### 4. 颜色相关属性

☐ chart.fillstyle

用于设置绘制折线区域所使用的填充颜色。

☐ chart.filled

用于设置是否填充折线区域。如果填充折线区域则会以指定的填充颜色来填充折线与水平坐标轴之间的区域。默认值为 false。

☐ chart.filled.range

当需要使用上下两根折线来绘制范围折线图时，需要将该属性设为 true。范围折线图需要使用两个数组作为绘制的数值来源。当指定填充颜色并进行填充时，只填充两根折线图之间的区域。默认值为 false。

### 5. 用户交互相关属性

☐ chart.crosshairs

该值被设为 true 时，将在当前光标位置处显示一个十字光标。默认值为 false。

☐ chart.crosshairs.linewidth

用于设置十字光标的线宽。默认值为 1。

☐ chart.crosshairs.color

用于设置十字光标的颜色。默认值为 #333。

❑ `chart.adjustable`

用于设置是否允许用户调整统计图。默认值为 `false`。

## 6. 标题相关属性

❑ `chart.title.xaxis`

设置水平坐标轴的标题文字。默认值为 `none`(不设置标题文字)。

❑ `chart.title.yaxis`

设置垂直坐标轴的标题文字。默认值为 `none`(不设置标题文字)。

❑ `chart.title.xaxis.pos`

用该值乘以 `chart.gutter` 属性中所设置的值得到水平坐标轴标题文字的位置。默认值为 0.25。

❑ `chart.title.yaxis.pos`

用该值乘以 `chart.gutter` 属性中所设置的值得到垂直坐标轴标题文字的位置。默认值为 0.25。

## 7. 格式相关属性

❑ `chart.scale.thousand`

用于指定数值的千位分隔符。默认值为 “,”。

❑ `chart.ymin`

用于设置 Y 坐标轴上数字的最小值。如果不指定, 则该值为 0。默认值为 `null`。

❑ `chart.ymax`

用于设置 Y 坐标轴上数字的最大值。如果不指定, 该值将被自动计算出来。默认值为 `null` (自动计算)。

## 8. 坐标轴相关属性

❑ `chart.tickdirection`

用于指定水平坐标轴上的刻度线是位于坐标轴的上方(该值为 1)还是下方(该值为 -1)。默认值为 -1。

❑ `chart.axis.color`

用于指定坐标轴颜色。默认值为 `black` (黑色)。

❑ `chart.xaxispos`

用于指定水平坐标轴的位置。可以指定的值为 `bottom` (底部) 或 `center` (中央)。默认值为 `bottom` (底部)。

❑ `chart.yaxispos`

用于指定垂直坐标轴的位置。可以指定的值为 `left` (统计图左侧) 或 `right` (统计图右侧)。默认值为 `left`。

❑ `chart.noaxes`

指定是否不绘制坐标轴。默认值为 `false` (绘制坐标轴)。

## 9. 其他属性

### ☐ chart.tickmarks

用于指定每段折线段之间的连接符号。可以设定的值为：

- dot (点)
  - circle (空心圆圈)
  - filledcircle (实心圆圈)
  - endcircle (只在整条折线的起点与终点处绘制空心圆圈)
  - square (空心方块)
  - endsquare (只在整条折线的起点与终点处绘制空心方块)
  - filledsquare (实心方块)
  - filledendsquare (只在整条折线的起点与终点处绘制实心方块)
  - tick (刻度线)
  - halftick (半刻度线)
  - endtick (只在整条折线的起点与终点处绘制刻度线)
  - cross (叉线)
  - borderedcircle (绘制效果与 dot 相同)
  - arrow (只在整条折线的终点处绘制一个箭头)
  - filledarrow (只在整条折线的终点处绘制一个实心箭头)
- 默认值为 null (没有连接符号)。

### ☐ chart.tickmarks.dot.color

当 chart.tickmarks 的属性值被指定为 dot 或 borderedcircle 时绘制点的颜色。默认值为 #fff。

### ☐ chart.stepped

用于设置是否将折线绘制为阶梯形式 (没有斜线, 只有竖线与横线)。默认值为 false。

### ☐ chart.chromefix

在 Google Chrome 浏览器中, 当一起使用阴影和晕开效果的时候, 存在一个很明显的 bug。通过将这个属性值设为 true 来修正该 bug。默认值为 true。

## 11.4.2 绘制基本折线图

接下来看一个绘制基本折线图的示例程序。该示例程序的功能与代码清单 11-1 的功能基本相同, 仍然是表现 2010 年常州第一百货公司长虹彩电的销售情况, 只不过在代码清单 11-1 中使用柱状图的形式来表现, 而在本示例程序中采用折线图的形式来进行表现。

该示例页面在浏览器中打开时的效果如图 11-4 所示。



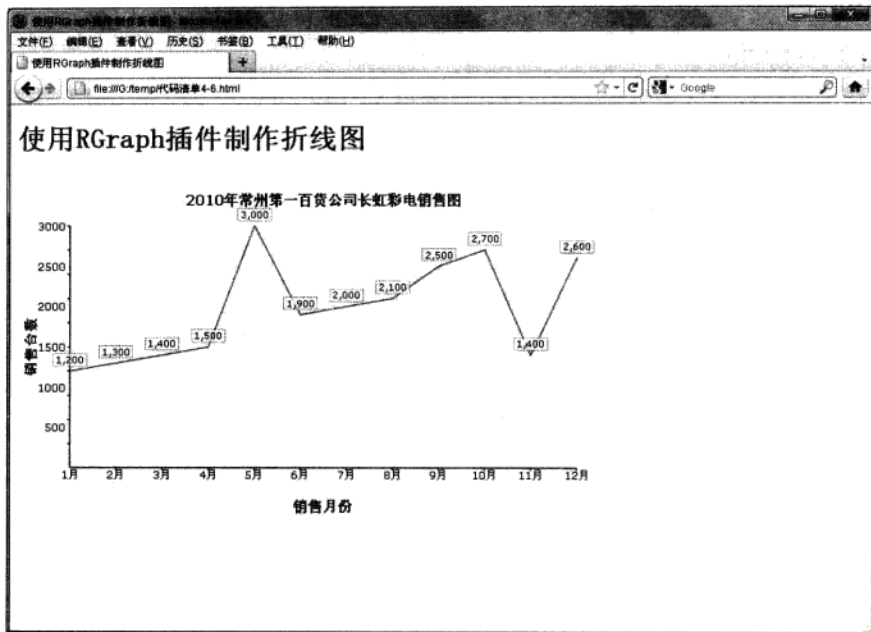


图 11-4 使用 RGraph 插件绘制基本折线图

该示例程序的完整代码如代码清单 11-6 所示。

代码清单 11-6 使用 RGraph 插件绘制基本折线图

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title> 使用 RGraph 插件制作折线图 </title>
<script src="RGraph.common.core.js"></script>
<script src="RGraph.line.js"></script>
<script src="RGraph.common.tooltips.js"></script>

<script>
function window_onload()
{
    // 绘制折线图, 指定数据
    var myGraph = new RGraph.Line('myCanvas', [1200,1300,1400,1500,3000,1900,
    2000,2100,2500,2700,1400,2600]);
    // 指定折线图标题
    myGraph.Set('chart.title', '2010 年常州第一百货公司长虹彩电销售图 ');
    // 指定 X 轴标题
    myGraph.Set('chart.title.xaxis', ' 销售月份 ');
    // 指定 Y 轴标题
    myGraph.Set('chart.title.yaxis', ' 销售台数 ');
```

```

// 指定 X 轴的坐标轴文字
myGraph.Set('chart.labels', ['1月','2月','3月','4月','5月','6月',
'7月','8月','9月','10月','11月','12月']);
// 指定 Y 轴的坐标轴文字
myGraph.Set('chart.ylabels.specific', ['3000','2500','2000','1500',
'1000','500']);
// 指定在折线连接点处绘制的说明销售数量的文字
myGraph.Set('chart.labels.above', true);
// 指定网格颜色
myGraph.Set('chart.background.grid.color', 'rgba(238,238,238,1)');
// 指定标签文字的绘制空间
myGraph.Set('chart.gutter', 60);
// 绘制折线图
myGraph.Draw();
}
</script>
</head>
<body onload="window.onload()">
<h1>使用 RGraph 插件制作折线图 </h1>
<canvas id="myCanvas" width="700" height="400">
    [您的浏览器不支持 canvas 元素]
</canvas>
</body>
</html>

```

### 11.4.3 使用 getPoint 方法

在绘制折线图时，可以通过 obj.getPoint 方法使开发者或用户知道哪个连接点获得了鼠标焦点。该方法返回一个数组，其中存放以下信息：

- 1) 用于绘制统计图对象的 canvas 元素。
- 2) 获得鼠标焦点的连接点在 X 轴上的坐标点。
- 3) 获得鼠标焦点的连接点在 Y 轴上的坐标点。
- 4) 获得鼠标焦点的连接点的序号（标示第几个连接点获得了鼠标焦点）。

下面来看一个使用该方法的示例。该示例在代码清单 11-6 的基础上稍加修改，当连接点获得鼠标焦点时鼠标指针会从指针形状变成手指形状，该连接点处也会突出显示一个蓝色实心圆圈，同时出现一个工具条提示信息，说明该连接点是水平坐标轴上哪一个绘制单位的连接点（在本示例的工具条提示中显示月份信息），如图 11-5 所示。

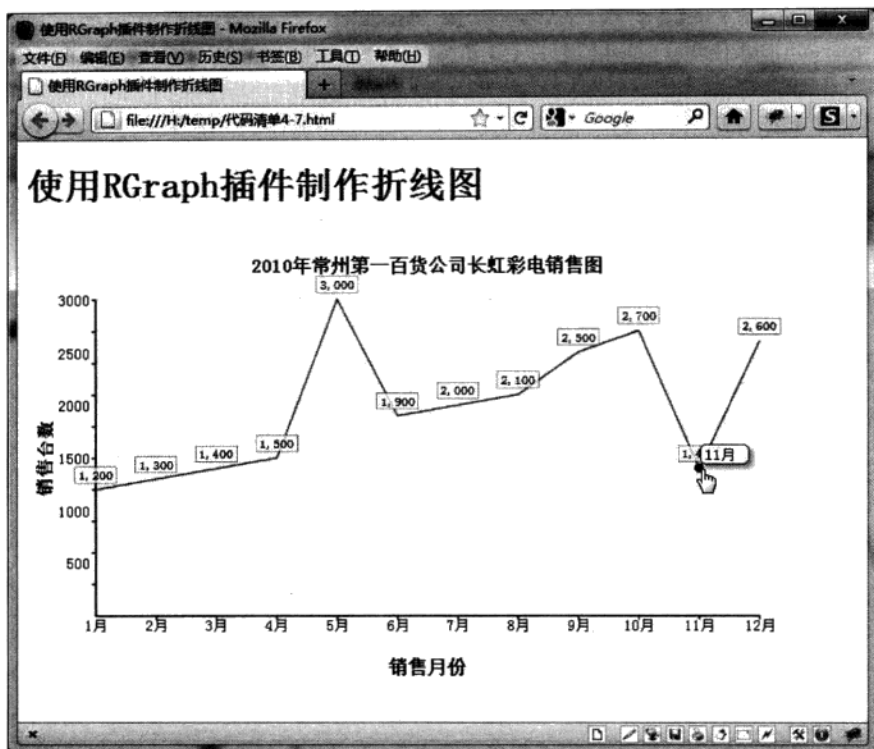


图 11-5 连接点获得鼠标焦点时出现工具条提示信息

这段示例程序的完整代码如代码清单 11-7 所示。

代码清单 11-7 连接点获得鼠标焦点时突出显示

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title> 使用 RGraph 插件制作折线图 </title>
<script src="RGraph.common.core.js"></script>
<script src="RGraph.line.js"></script>
<script src="RGraph.common.tooltips.js"></script>

<script>
function window_onload()
{
    // 绘制折线图, 指定数据
    var myGraph = new RGraph.Line('myCanvas', [1200,1300,1400,1500,3000,1900,
    2000,2100,2500,2700,1400,2600]);
    // 指定折线图标题
    myGraph.Set('chart.title', '2010 年常州第一百货公司长虹彩电销售图 ');
```

```

// 指定 X 轴标题
myGraph.Set('chart.title.xaxis', '销售月份');
// 指定 Y 轴标题
myGraph.Set('chart.title.yaxis', '销售台数');
// 指定 X 轴的坐标轴文字
myGraph.Set('chart.labels', ['1月', '2月', '3月', '4月', '5月', '6月',
'7月', '8月', '9月', '10月', '11月', '12月']);
// 指定 Y 轴的坐标轴文字
myGraph.Set('chart.ylabels.specific', ['3000', '2500', '2000', '1500',
'1000', '500']);
// 指定在折线连接点处绘制说明销售数量的文字
myGraph.Set('chart.labels.above', true);
// 指定网格颜色
myGraph.Set('chart.background.grid.color', 'rgba(238,238,238,1)');
// 指定标签文字的绘制空间
myGraph.Set('chart.gutter', 60);
// 绘制折线图
myGraph.Draw();
// 注册控件
RGraph.Register(myGraph);
// 书写鼠标在折线图上移动时的函数
myGraph.canvas.onmousemove = function (e)
{
    // 注册事件
    RGraph.FixEventObject(e);

    var canvas = e.target; // 获得绘制折线图的 canvas 元素

    // 获得绘制折线图的 canvas 元素的图形上下文对象
    var context = canvas.getContext('2d');
    var obj = e.target.__object__; // 获得事件对象

    // 使用 getPoint 方法来得到取得光标焦点的连接点
    var point = obj.getPoint(e);

    if (point) // 如果存在取得光标焦点的连接点
    {
        canvas.style.cursor = 'pointer'; // 改变鼠标指针为手指形状

        // 如果工具条提示已经被显示
        if (RGraph.Registry.Get('chart.tooltip')
        && RGraph.Registry.Get('chart.tooltip').__index__ == point[3]) {
            return;
        }
    }

    // 重绘折线图

```

```

    RGraph.Redraw();

    // 显示工具条提示
    RGraph.Tooltip(canvas, obj.Get('chart.labels')[point[3]],
        e.pageX, e.pageY, point[3]);

    // 突出显示连接点
    context.fillStyle = 'blue'; // 使用蓝色填充
    context.beginPath(); // 开始创建路径
    context.moveTo(point[1], point[2]); // 将绘制起点移动到连接点上
    context.arc(point[1], point[2], 4, 0, 6.26, 0); // 创建圆形路径
    context.fill(); // 填充圆圈

    return;
}
canvas.style.cursor = 'default'; // 恢复默认鼠标指针
}
}
// 在其他位置处单击时取消当前显示的工具条提示信息及蓝色放大折线点
window.onclick = function ()
{
    RGraph.Redraw();
}
</script>
</head>
<body onload="window.onload()">
<h1> 使用 RGraph 插件制作折线图 </h1>
<canvas id="myCanvas" width="700" height="400">
    [ 您的浏览器不支持 canvas 元素 ]
</canvas>
</body>
</html>

```

#### 11.4.4 在一个折线图中绘制多根折线

接下来看一个在一个折线图中绘制多根折线的示例程序。该示例程序的功能与代码清单 11-3 中的功能基本相同，仍然是表现 2010 年常州第一百货公司长虹彩电与康佳彩电的销售情况，只不过在代码清单 11-3 中使用柱状图的形式来表现，而在本示例程序中采用折线图的形式来进行表现。

该示例页面在浏览器中的显示效果如图 11-6 所示。

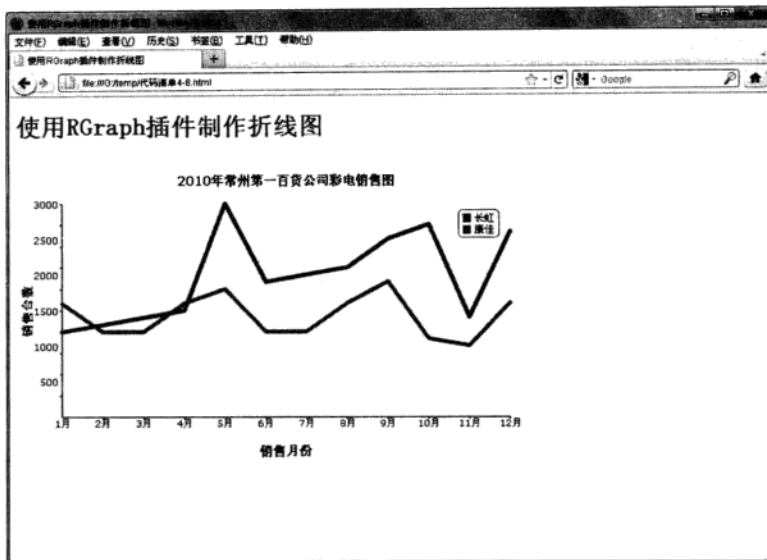


图 11-6 在一个折线图中绘制多根折线

该示例程序的完整代码如代码清单 11-8 所示。

代码清单 11-8 在一个折线图中绘制多根折线

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title> 使用 RGraph 插件制作折线图 </title>
<script src="RGraph.common.core.js"></script>
<script src="RGraph.line.js"></script>
<script src="RGraph.common.tooltips.js"></script>

<script>
function window_onload()
{
    // 绘制折线图，指定数据
    myGraph = new RGraph.Line('myCanvas', [1200,1300,1400,1500,3000,1900,
        2000,2100,2500,2700,1400,2600], [1600,1200,1200,1600,1800,1200,1200,
        1600,1900,1100,1000,1600]);
    // 指定折线图标题
    myGraph.Set('chart.title', '2010 年常州第一百货公司彩电销售图 ');
    // 指定 X 轴标题
    myGraph.Set('chart.title.xaxis', '销售月份');
    // 指定 Y 轴标题
    myGraph.Set('chart.title.yaxis', '销售台数');
    // 指定折线图例被绘制在图例区域中
    myGraph.Set('chart.key.position', 'graph');
    // 指定图例文字
```

```

myGraph.Set('chart.key', ['长虹', '康佳']);
// 指定折线颜色
myGraph.Set('chart.colors', ['blue', 'green']);
// 指定 X 轴的坐标轴文字
myGraph.Set('chart.labels', ['1月', '2月', '3月', '4月', '5月', '6月',
'7月', '8月', '9月', '10月', '11月', '12月']);
// 指定 Y 轴的坐标轴文字
myGraph.Set('chart.ylabels.specific', ['3000', '2500', '2000', '1500',
'1000', '500']);
// 指定线宽
myGraph.Set('chart.linewidth', 5);
// 指定网格颜色
myGraph.Set('chart.background.grid.color', 'rgba(238,238,238,1)');
// 指定标签文字的绘制空间
myGraph.Set('chart.gutter', 60);
// 绘制折线图
myGraph.Draw();
}
</script>
</head>
<body onload="window.onload()">
<h1>使用 RGraph 插件制作折线图 </h1>
<canvas id="myCanvas" width="700" height="400">
    [您的浏览器不支持 canvas 元素]
</canvas>
</body>
</html>

```

### 11.4.5 绘制范围折线图

绘制范围折线图，是指绘制上下两根折线，上部的折线用于反映统计范围内的峰值，下部的折线用于反映统计范围内的谷值，然后用某种颜色填充上下两根折线之间的封闭区域，使人能够一目了然每一个绘制单位（例如每个月）中某个统计值的峰值与谷值情况。

接下来看一个示例程序，该程序反映 2010 年常州第一百货公司长虹彩电的销售情况，上部的折线反映每个月中该彩电的销售峰值（最多一天能卖掉多少台），下部的折线反映每月的销售谷值（最少一天能卖掉多少台）。

该示例的页面在浏览器中的显示效果如图 11-7 所示。

制作范围折线图的关键在于在统计图中使用如下几个属性：

❑ 在指定数据来源时使用两个数组来分别指定峰值数组与谷值数组，代码如下所示。

```

myGraph = new RGraph.Line('myCanvas',
[160,130,140,160,300,190,200,210,250,270,140,260],
[120,120,120,150,180,120,120,160,190,110,100,160]);

```

❑ 使用 chart.colors 属性设置上下两根折线的折线颜色，代码如下所示。

```

myGraph.Set('chart.colors', ['blue']);

```

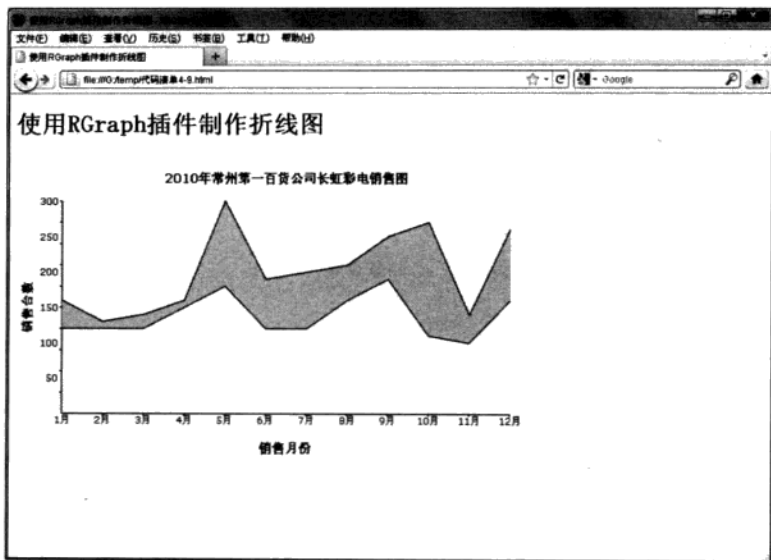


图 11-7 使用 RGraph 插件制作范围折线图

□ 使用 `chart.fillstyle` 属性设置上下两根折线区域之间的填充颜色，代码如下所示。

```
myGraph.Set('chart.fillstyle', 'lightgreen');
```

□ 将 `chart.filled` 属性的属性值设为 `true`，表示用指定的填充颜色填充折线内部区域，代码如下所示。

```
myGraph.Set('chart.filled', true);
```

□ 将 `chart.filled.range` 属性的属性值设为 `true`，表示只填充上下两根折线之间的区域，代码如下所示。

```
myGraph.Set('chart.filled.range', true);
```

该示例程序的完整代码如代码清单 11-9 所示。

代码清单 11-9 绘制范围折线图

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title> 使用 RGraph 插件制作折线图 </title>
<script src="RGraph.common.core.js"></script>
<script src="RGraph.line.js"></script>
<script src="RGraph.common.tooltips.js"></script>

<script>
function window_onload()
{
```



```

// 绘制折线图, 指定数据
myGraph = new RGraph.Line('myCanvas',
    [160,130,140,160,300,190,200,210,250,270,140,260],
    [120,120,120,150,180,120,120,160,190,110,100,160]);
// 指定折线图标题
myGraph.Set('chart.title','2010 年常州第一百货公司长虹彩电销售图');
// 指定 X 轴标题
myGraph.Set('chart.title.xaxis','销售月份');
// 指定 Y 轴标题
myGraph.Set('chart.title.yaxis','销售台数');
// 指定折线颜色
myGraph.Set('chart.colors', ['blue']);
// 指定 X 轴的坐标轴文字
myGraph.Set('chart.labels', ['1 月','2 月','3 月','4 月','5 月','6 月',
    '7 月','8 月','9 月','10 月','11 月','12 月']);
// 指定 Y 轴的坐标轴文字
myGraph.Set('chart.ylabels.specific',['300','250','200','150','100',
    '50']);
// 指定折线区域内的填充颜色
myGraph.Set('chart.fillstyle', 'lightgreen');
// 用指定的填充颜色填充折线内部区域
myGraph.Set('chart.filled',true);
// 指定只填充上下两根折线之间的区域
myGraph.Set('chart.filled.range',true);
// 指定网格颜色
myGraph.Set('chart.background.grid.color', 'rgba(238,238,238,1)');
// 指定标签文字的绘制空间
myGraph.Set('chart.gutter', 60);
// 绘制折线图
myGraph.Draw();
}
</script>
</head>
<body onload="window.onload()">
<h1> 使用 RGraph 插件制作折线图 </h1>
<canvas id="myCanvas" width="700" height="400">
    [ 您的浏览器不支持 canvas 元素 ]
</canvas>
</body>
</html>

```

### 11.4.6 在一个折线图中使用左右两根不同统计单位的垂直坐标轴

接下来看一个在左右两根垂直坐标轴上分别使用两种统计单位绘制折线图的示例。该示例仍然反映 2010 年常州第一百货公司长虹彩电的销售情况, 但这一次在折线图中绘制两根折线, 分别体现长虹彩电的销售台数与销售成本, 左边的垂直坐标轴用来标注长虹彩电的销售台数, 顶部最大刻度为 3000 台, 右边的垂直坐标轴用来标注销售成本, 顶部最大刻度为

20 000 元。

该示例的页面在浏览器中的显示效果如图 11-8 所示。

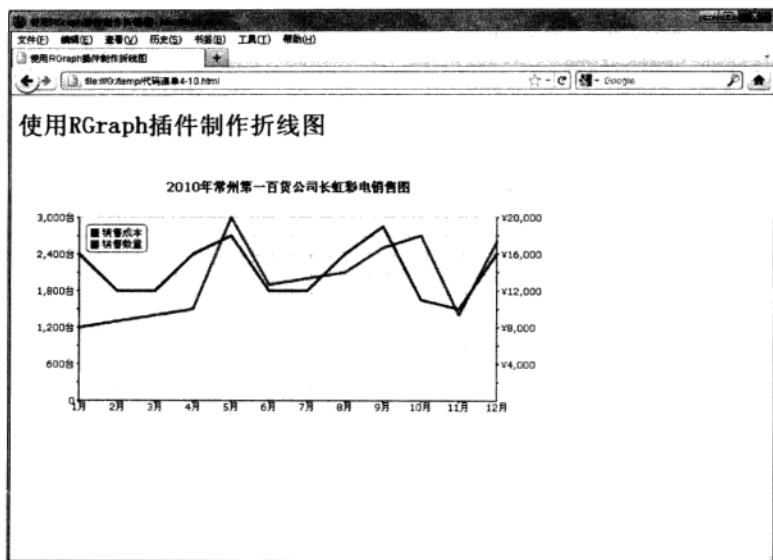


图 11-8 在一个折线图中使用左右两根不同单位的垂直坐标轴

在同一折线图中绘制两种单位的垂直坐标轴的关键在于，在同一个 canvas 元素中绘制两个折线图，然后通过属性设置使其看上去是一个折线图。该示例程序的完整代码如代码清单 11-10 所示。

代码清单 11-10 在一个折线图中使用左右两根不同单位的垂直坐标轴

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title> 使用 RGraph 插件制作折线图 </title>
<script src="RGraph.common.core.js"></script>
<script src="RGraph.line.js"></script>
<script src="RGraph.common.tooltips.js"></script>

<script>
function window_onload()
{
    // 绘制销售数量折线图，指定数据
    line1= new RGraph.Line('myCanvas',[1200,1300,1400,1500,3000,1900,2000,
    2100,2500,2700,1400,2600]);
    // 绘制统计图标题
    line1.Set('chart.title', '2010 年常州第一百货公司长虹彩电销售图 ');
    // 指定不绘制水平坐标轴
    line1.Set('chart.noaxis', true);
```

```

// 指定左侧垂直坐标轴上标签文字的后缀单位
line1.Set('chart.units.post', '台');
// 指定折线图线宽
line1.Set('chart.linewidth', 3);
// 指定左侧垂直坐标轴的顶部最大数字
line1.Set('chart.ymax', 3000);
// 指定标签文字的绘制空间
line1.Set('chart.gutter', 80);
// 绘制销售数量折线图
line1.Draw();

// 绘制销售成本折线图
line2 = new RGraph.Line('myCanvas', [16000, 12000, 12000, 16000, 18000, 12000,
12000, 16000, 19000, 11000, 10000, 16000]);
// 指定 X 轴的坐标轴文字
line2.Set('chart.labels', ['1月', '2月', '3月', '4月', '5月', '6月', '7月',
'8月', '9月', '10月', '11月', '12月']);
// 指定折线与图例中的颜色
line2.Set('chart.colors', ['blue', 'red']);
// 指定图例文字
line2.Set('chart.key', ['销售成本', '销售数量']);
// 指定销售成本折线图的垂直坐标轴位于右侧
line2.Set('chart.yaxispos', 'right');
// 指定右侧垂直坐标轴的顶部最大数字
line2.Set('chart.ymax', 20000);
// 指定右侧垂直坐标轴上标签文字的前缀单位
line2.Set('chart.units.pre', '¥');
// 指定标签文字的绘制空间
line2.Set('chart.gutter', 80);
// 指定线宽
line2.Set('chart.linewidth', 3);
// 指定不绘制网格
line2.Set('chart.background.grid', false);
// 绘制销售成本折线图
line2.Draw();
}
</script>

</head>
<body onload="window.onload()">
<h1>使用 RGraph 插件制作折线图 </h1>
<canvas id="myCanvas" width="700" height="400">
    [您的浏览器不支持 canvas 元素]
</canvas>
</body>
</html>

```

### 11.4.7 在一个统计图中绘制柱状图与折线图

接下来看一个在统计图中同时绘制柱状图与折线图的示例程序，该示例程序的功能与代

码清单 11-1 和代码清单 11-6 中的功能大致相同，都是反映 2010 年常州第一百货公司长虹与康佳两种彩电的销售情况，使用柱状图来体现长虹彩电每个月的销售数量，使用折线图来体现康佳彩电每个月的销售数量。

该示例的页面在浏览器中的显示效果如图 11-9 所示。

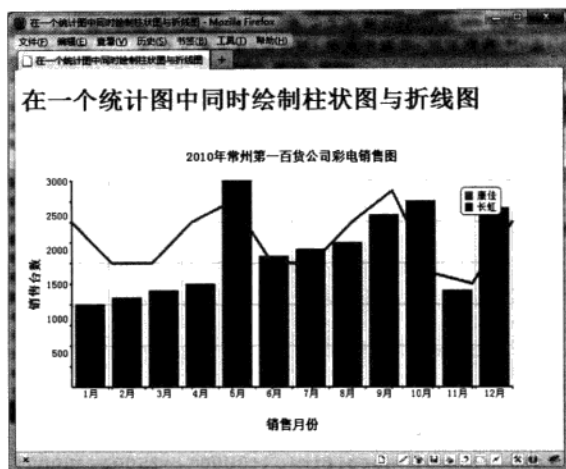


图 11-9 在一个统计图中同时绘制柱状图与折线图

在一个统计图中同时绘制柱状图与折线图的关键也在于，在同一个 canvas 元素中绘制一个柱状图与一个折线图，然后通过属性设置使其看上去为一个统计图。该示例程序的完整代码如代码清单 11-11 所示。

代码清单 11-11 在一个统计图中同时绘制柱状图与折线图

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title> 在一个统计图中同时绘制柱状图与折线图 </title>
<script src="RGraph.common.core.js"></script>
<script src="RGraph.bar.js"></script>
<script src="RGraph.line.js"></script>
</script>
function window_onload()
{
    // 绘制柱状图，指定数据
    myBar = new RGraph.Bar('myCanvas', [1200,1300,1400,1500,3000,1900,2000,
    2100,2500,2700,1400,2600]);
    // 指定统计图标题
    myBar.Set('chart.title', '2010 年常州第一百货公司彩电销售图');
    // 指定 X 轴标题
    myBar.Set('chart.title.xaxis', '销售月份');
    // 指定 Y 轴标题
```

```

myBar.Set('chart.title.yaxis','销售台数');
// 指定 X 轴的坐标轴文字
myBar.Set('chart.labels',['1月','2月','3月','4月','5月','6月','7月',
'8月','9月','10月','11月','12月']);
// 指定 Y 轴的坐标轴文字
myBar.Set('chart.ylabels.specific',['3000','2500','2000','1500','1000',
'500']);
// 指定网格自动与坐标轴单位对齐
myBar.Set('chart.background.grid.autofit', true);
myBar.Set('chart.background.grid.autofit.align', true);
// 指定标签文字所使用的空间尺寸
myBar.Set('chart.gutter',65);
myBar.Draw();

// 绘制折线图, 指定数据
var myLine = new RGraph.Line('myCanvas',[1600,1200,1200,1600,1800,1200,
1200,1600,1900,1100,1000,1600]);
// 指定折线与图例中的颜色
myLine.Set('chart.colors',['red','blue']);
// 指定图例文字
myLine.Set('chart.key',['康佳','长虹']);
// 指定 Y 轴的坐标轴文字
myLine.Set('chart.ylabels.specific',['3000','2500','2000','1500',
'1000','500']);
// 指定线宽
myLine.Set('chart.linewidth', 3);
// 指定不绘制网格
myLine.Set('chart.background.grid', false);
// 指定标签文字的绘制空间
myLine.Set('chart.gutter', 65);
// 绘制折线图
myLine.Draw();
}

</script>
</head>
<body onload="window.onload()">
<h1> 在一个统计图中同时绘制柱状图与折线图 </h1>
<canvas id="myCanvas" width="700" height="400">
    [ 您的浏览器不支持 canvas 元素 ]
</canvas>
</body>
</html>

```

### 11.4.8 绘制动态折线图

再看一个绘制实时更新的动态折线图的示例程序。在实际应用中, 可以通过 AJAX 等方法从后台取得服务器端的动态数据来实时更新折线图。在本示例程序中, 只采用每 250 毫

秒产生一个随机数的方法来模拟动态数据。这种实时更新的折线图可以应用在反映网络带宽、频率和实时用电信息等很多需要根据现场情况进行实时更新的统计图中。

本示例页面在浏览器中的显示效果如图 11-10 所示（注意：每 250 毫秒折线图会更新一次）。

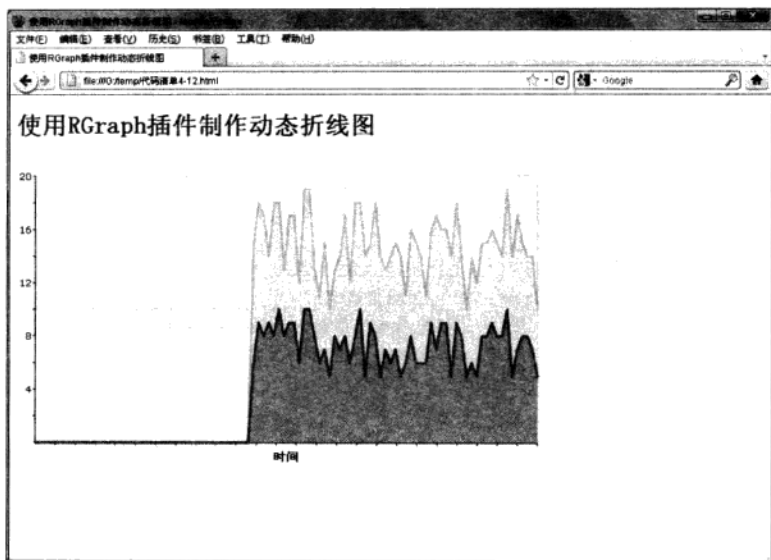


图 11-10 使用 RGraph 插件绘制动态折线图

这个示例程序的完整代码如代码清单 11-12 所示。

代码清单 11-12 绘制动态折线图

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title> 使用 RGraph 插件制作动态折线图 </title>
<script src="RGraph.common.core.js"></script>
<script src="RGraph.line.js"></script>
<script>
function window_onload()
{
    d1 = []; // 存放上部折线图使用数据的数组
    d2 = []; // 存放下部折线图使用数据的数组

    // 用 null 值填充数组
    for (var i=0; i< 100; ++i)
    {
        d1.push(null);
        d2.push(null);
    }
    // 绘制折线图
```

```

        drawGraph();
    }

    // 设置折线图属性
    function getGraph(id, d1, d2)
    {
        var graph = new RGraph.Line(id, d1, d2); // 获取折线图数据
        graph.Set('chart.gutter', 25); // 设置标签文字使用空间
        graph.Set('chart.title.xaxis', '时间'); // 设置水平坐标轴标题
        graph.Set('chart.filled', true); // 使用填充色填充折线区域

        // 指定上部折线区域与下部折线区域的填充色
        graph.Set('chart.fillstyle', ['#daflfa', '#faa']);
        // 指定上部折线与下部折线的颜色
        graph.Set('chart.colors', ['rgb(169, 222, 244)', 'red']);
        graph.Set('chart.linewidth', 3); // 指定线宽
        graph.Set('chart.ymax', 20); // 指定垂直坐标轴上的最大数值
        graph.Set('chart.xticks', 25); // 指定水平坐标轴上的刻度线
        return graph; // 返回设置好的折线图
    }

    // 绘制折线图
    function drawGraph (e)
    {
        // 清除之前绘制的折线图
        RGraph.Clear(document.getElementById("myCanvas"));

        var graph = getGraph('myCanvas', d1, d2); // 获取设置好属性的折线图
        graph.Draw(); // 绘制折线图

        // 使用随机数字填充折线图所使用的数据数组
        d1.push(RGraph.random(5, 10));
        d2.push(RGraph.random(5, 10));

        // 如果数组已经填满, 则移出数组中最前面的数字, 并将数组中每个数字向前移位
        if (d1.length > 100) {
            d1 = RGraph.array_shift(d1);
            d2 = RGraph.array_shift(d2);
        }

        // 设置每 250 毫秒更新折线图
        setTimeout(drawGraph, 250);
    }
</script>
</head>
<body onload="window_onload()">
<h1>使用 RGraph 插件制作动态折线图 </h1>
<canvas id="myCanvas" width="700" height="400">
    [ 您的浏览器不支持 canvas 元素 ]
</canvas>
</body>
</html>

```

---

## 11.5 绘制饼图

### 11.5.1 绘制饼图时所用到的属性

本节具体介绍一下如何使用 RGraph 插件绘制饼图。在讲解示例前，先来了解一下在绘制饼图时所使用到的属性。

#### ☐ chart.strokeStyle

用来指定饼块与饼块之间分隔线的颜色。使用背景色来设置该属性值，并且将线宽设置为 5，可以使饼图具有分离效果。默认值为 #999。

#### ☐ chart.labels.sticks

用来设置是否在饼块与旁边的说明文字之间绘制一根短的连接线。默认值为 false（不绘制）。

#### ☐ chart.labels.sticks.color

用来指定饼块与旁边的说明文字之间短连接线的颜色。默认值为 #aaa。

#### ☐ chart.linewidth

将该值设定为 5，将 chart.strokeStyle 的属性值设定为与背景色相同的值，可以使饼图具有分离效果。默认值为 1。

#### ☐ chart.variant

可以设定的值为 pie 与 donut。将该值设定为 donut 时将绘制环形饼图。默认值为 pie。

### 11.5.2 示例程序

接下来介绍一个绘制饼图的示例程序，该示例程序表现的是 2010 年常州第一百货公司各种彩电销售数量的分布情况。该示例程序在页面打开时的效果如图 11-11 所示。

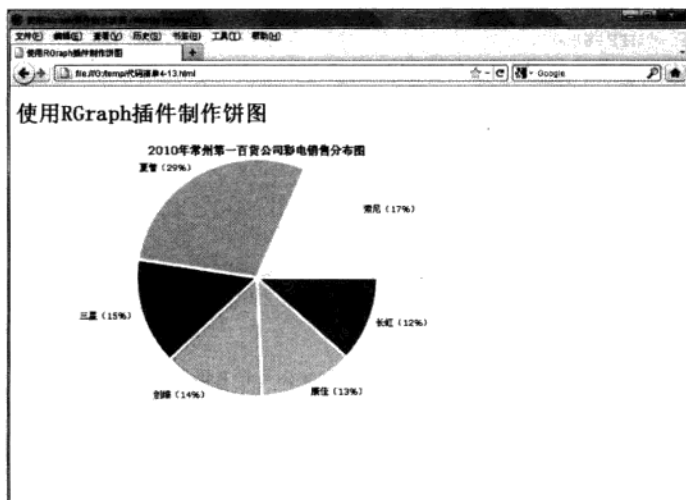


图 11-11 使用 RGraph 插件制作饼图



该示例程序中使用了工具条提示功能，当鼠标指针移动到某个饼块上时，这个饼块会呈现一个 3D 效果的突出显示，并且在该饼块上出现一个工具条提示信息，如图 11-12 所示。

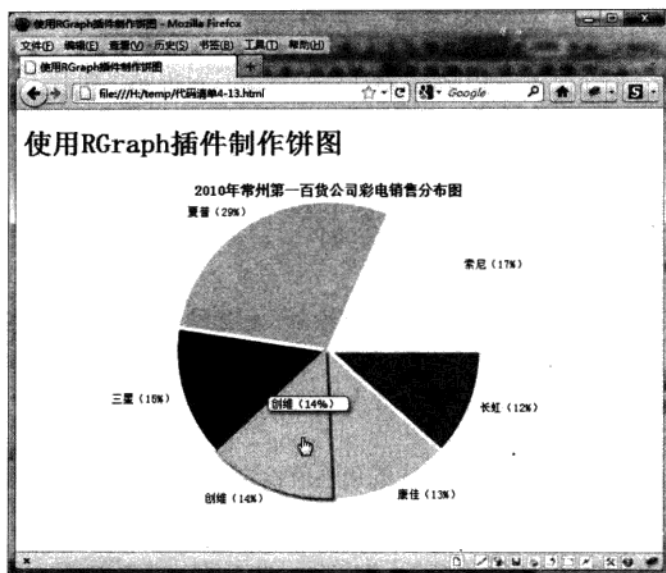


图 11-12 鼠标在饼块上移动时显示 3D 效果和工具条提示信息

这段示例程序的完整代码如代码清单 11-13 所示。

代码清单 11-13 使用 RGraph 插件制作饼图

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title> 使用 RGraph 插件制作饼图 </title>
<script src="RGraph.common.core.js"></script>
<script src="RGraph.pie.js"></script>
<script src="RGraph.common.tooltips.js"></script>
<script>
function window_onload()
{
    // 绘制饼图，获取饼图数据
    var pie=new RGraph.Pie('myCanvas',[12000,13000,14000,15000,30000,
    19000]);
    // 绘制饼图标题
    pie.Set('chart.title', '2010 年常州第一百货公司彩电销售分布图');
    // 绘制饼图标签文字
    pie.Set('chart.labels', ['长虹 (12%)', '康佳 (13%)', '创维 (14%)',
    '三星 (15%)', '夏普 (29%)', '索尼 (17%)']);
    // 指定饼图分隔线宽
    pie.Set('chart.linewidth', 5);
```

```

// 指定饼图分隔线颜色
pie.Set('chart.strokestyle','white');
// 指定工具条提示信息的出现效果为淡入效果
pie.Set('chart.tooltips.effect','fade');
// 指定当鼠标指针在饼块上移动时出现工具条提示信息
pie.Set('chart.tooltips.event','onmousemove');
// 指定工具条提示信息的文字
pie.Set('chart.tooltips', ['长虹 (12%)','康佳 (13%)','创维 (14%)',
'三星 (15%)','夏普 (29%)','索尼 (17%)']);
// 指定工具条提示信息具有 3d 效果
pie.Set('chart.highlight.style','3d');
// 绘制饼图
pie.Draw();
}
</script>
</head>
<body onload="window.onload()">
<h1>使用 RGraph 插件制作饼图 </h1>
<canvas id="myCanvas" width="700" height="400">
    [您的浏览器不支持 canvas 元素]
</canvas>
</body>
</html>

```

在这段示例代码中，使用在每个饼块旁添加说明文字的方法来对每个饼块所代表的含义进行说明。也可以使用图例的方法来对每个饼块进行说明，例如将代码清单 11-13 中的代码进行如下修改，修改后重新在浏览器中打开该页面，显示效果如图 11-13 所示。

```

// 添加使用图例
pie.Set('chart.key', ['长虹 (12%)','康佳 (13%)','创维 (14%)',
'三星 (15%)','夏普 (29%)','索尼 (17%)']);
// 设置图例背景色为白色
pie.Set('chart.key.background','white');
// 删除每个饼块旁边的说明文字
//pie.Set('chart.labels', ['长虹 (12%)','康佳 (13%)','创维 (14%)',
'三星 (15%)','夏普 (29%)','索尼 (17%)']);

```

另外，只需在饼图中添加使用 `chart.variant` 属性，并将该属性值设为“donut”（代码如下所示），即可绘制出环形饼图，如图 11-14 所示。

```

// 绘制环形饼图
pie.Set('chart.variant','donut');

```

### 11.5.3 使用 getSegment 方法

在绘制饼图时，可以通过 `obj.getSegment` 方法使开发者或用户知道哪个饼块被单击。该方法返回一个数组，其中存放以下信息：

- 1) 获得被单击饼块在 X 轴上的坐标点。

- 2) 获得被单击饼块在 Y 轴上的坐标点。
- 3) 获得被单击饼块的绘制半径。
- 4) 获得被单击饼块的起始绘制角度。
- 5) 获得被单击饼块的结束绘制角度。

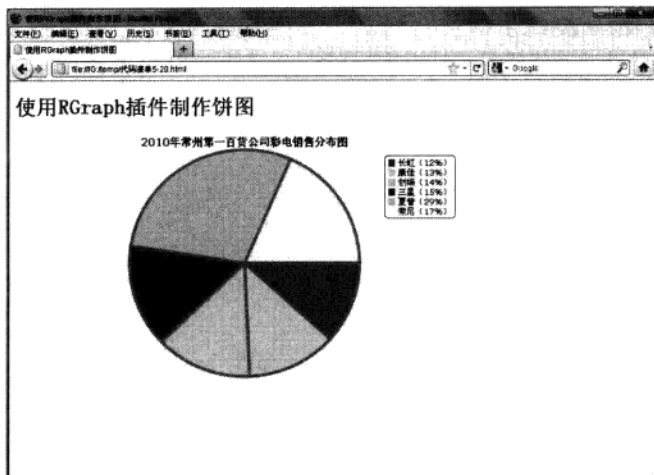


图 11-13 使用图例来对每个饼块的含义进行说明

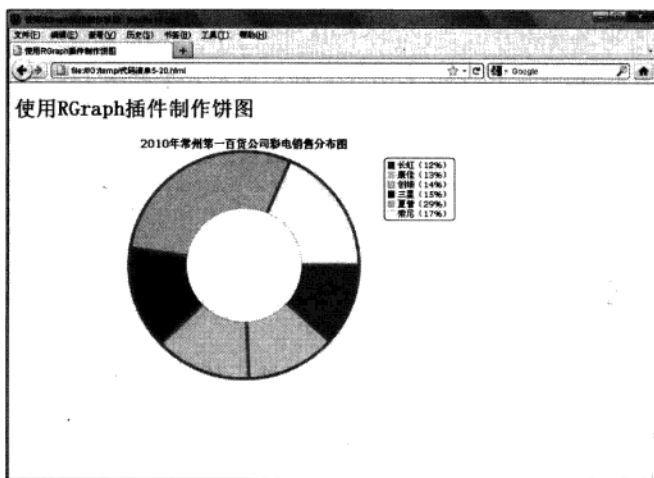


图 11-14 绘制环形饼图

下面介绍一个使用 `getSegment` 方法的示例。该示例的程序在代码清单 11-13 上稍加修改，当饼块被单击时会在被单击饼块之上再绘制一个相同尺寸的白色半透明饼块，使被点击饼块呈现一种白色半透明的效果，如图 11-15 所示。

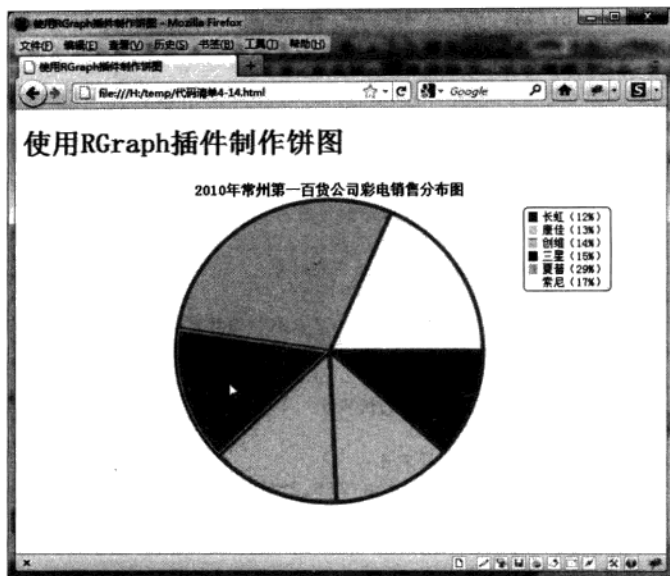


图 11-15 饼块被单击时呈现白色半透明效果

这段示例程序的完整代码如代码清单 11-14 所示。

代码清单 11-14 饼块被单击时呈现白色半透明效果

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title> 使用 RGraph 插件制作饼图 </title>
<script src="RGraph.common.core.js"></script>
<script src="RGraph.pie.js"></script>
<script src="RGraph.common.tooltips.js"></script>

<script>
function window_onload()
{
    var pie=new RGraph.Pie('myCanvas', [12000,13000,14000,15000,30000,
    19000]);
    // 绘制饼图标题
    pie.Set('chart.title', '2010 年常州第一百货公司彩电销售分布图');
    // 绘制饼图图例
    pie.Set('chart.key', [ '长虹 (12%)', '康佳 (13%)', '创维 (14%)',
    '三星 (15%)', '夏普 (29%)', '索尼 (17%)' ]);
    // 指定图例背景色
    pie.Set('chart.key.background', 'white');
    // 指定饼块间的分隔线颜色
    pie.Set('chart.linewidth', 5);
    // 绘制饼图
    pie.Draw();
    // 注册控件
```

```

RGraph.Register(pie);
// 指定饼图被单击时的函数
pie.canvas.onclick = function (e)
{
    RGraph.FixEventObject(e); // 注册事件
    RGraph.Redraw(); // 重绘饼图

    var canvas = e.target; // 获取绘制饼图的 canvas 元素

    // 获取绘制饼图的 canvas 元素的图形上下文对象
    var context = canvas.getContext('2d');
    var obj = canvas.__object__; // 获取饼图对象
    var segment = obj.getSegment(e); // 获取被单击的饼块

    if (segment) // 如果存在被单击的饼块
    {
        // 指定白色半透明颜色为填充色
        context.fillStyle = 'rgba(255,255,255,0.5)';
        context.beginPath(); // 开始创建路径

        // 将角度转换为半径
        segment[3] /= 57.29;
        segment[4] /= 57.29;

        // 将绘制起点移动到被单击的饼块处
        context.moveTo(segment[0], segment[1]);
        // 在被单击的饼块上再绘制相同尺寸的饼块
        context.arc(segment[0], segment[1], segment[2], segment[3],
            segment[4], 0);
        context.stroke(); // 绘制饼块边框
        context.fill(); // 填充饼块

        e.stopPropagation(); // 阻止事件传播
    }
}

// 指定页面被单击时的函数
window.onclick = function (e)
{
    RGraph.Redraw(); // 重绘饼图
}
</script>
</head>
<body onload="window.onload()">
<h1>使用 RGraph 插件制作饼图 </h1>
<canvas id="myCanvas" width="700" height="400">
    [ 您的浏览器不支持 canvas 元素 ]
</canvas>
</body>
</html>

```

## 11.6 绘制横向柱状图

本节具体介绍如何使用 RGraph 插件绘制横向柱状图。在讲解实例前，先来看一下在绘制横向柱状图时所用到的属性。

### 11.6.1 绘制横向柱状图时所用到的属性

除了前面所讲的绘制所有统计图时用到的属性外，绘制横向柱状图时还会用到如下属性。

#### 1 背景相关属性

❑ `chart.background.barcolor1`

设置柱状图背景网格中的奇数行（从上往下数）的背景颜色（默认为白色）。

❑ `chart.background.barcolor2`

设置柱状图背景网格中的偶数行（从上往下数）的背景颜色（默认为白色）。

❑ `chart.background.grid`

指定是否显示背景网格。默认值为 `true`（显示）。

❑ `chart.background.grid.color`

设置背景网格线的颜色。默认颜色为 `#ddd`。

❑ `chart.background.grid.hsize`

设置背景网格的每格宽度。默认值为 40。

❑ `chart.background.grid.vsize`

设置背景网格的每格高度。默认值为 18。

❑ `chart.background.grid.width`

设置背景网格线宽。允许属性值不为整数（例如 0.5）。默认值为 1。

❑ `chart.background.grid.border`

设置是否在背景网格周围绘制一个边框线。默认值为 `true`。

❑ `chart.background.grid.hlines`

设置是否绘制背景网格中的水平线。默认值为 `true`。

❑ `chart.background.grid.vlines`

设置是否绘制背景网格中的垂直线。默认值为 `true`。

❑ `chart.background.grid.autofit`

不指定背景网格中每格宽度或高度，只指定绘制多少条垂直线与水平线，然后采用自动计算方式。默认值为 `false`（不采用这种方式）。

❑ `chart.background.grid.autofit.numhlines`

指定当使用自动计算方式绘制背景网格时需要绘制的水平线条数。默认值为 7。

❑ `chart.background.grid.autofit.numvlines`

指定当使用自动计算方式绘制背景网格时需要绘制的垂直线条数。默认值为 20。

❑ `chart.background.grid.autofit.align`

指定背景网格线是否自动与坐标轴刻度对齐。默认值为 `false`。

## 2. 标签相关属性

❑ `chart.xlabels`

用于设置是否绘制水平坐标轴。默认值为 `true`。

❑ `chart.labels.above`

该值为 `true` 时，将会在每根柱子的右端显示一个具体说明该柱子所代表数值的数字。默认值为 `false`。

❑ `chart.labels.above.decimals`

用于设置坐标轴每根柱子右端数值精度（允许精确到小数点后几位）。默认值为 `0`。

## 3. 标题相关属性

❑ `chart.title.xaxis`

设置水平坐标轴的标题文字。默认值为 `none`（不设置标题文字）。

❑ `chart.title.yaxis`

设置垂直坐标轴的标题文字。默认值为 `none`（不设置标题文字）。

❑ `chart.title.xaxis.pos`

用该值乘以在 `chart.gutter` 属性中所设置的值得到水平坐标轴标题文字的位置。默认值为 `0.25`。

❑ `chart.title.yaxis.pos`

用该值乘以在 `chart.gutter` 属性中所设置的值得到垂直坐标轴标题文字的位置。默认值为 `0.25`。

## 4. 格式与坐标轴相关属性

❑ `chart.scale.thousand`

用于指定数值的千位分隔符。默认值为 “,”。

❑ `chart.xmax`

用于设置水平坐标轴上数字的最大值。如果不指定，则该值将被自动计算出来。默认值为 `null`。（自动计算）

❑ `chart.xmin`

用于设置水平坐标轴上数字的最小值。默认值为 `0`。

❑ `chart.yaxispos`

用于指定垂直坐标轴的位置。可以指定的值为 `left`（左端）或 `center`（中央）。默认值为 `left`。

❑ `chart.axis.color`

用于指定坐标轴颜色。默认值为 `black`（黑色）。

## 5. 其他相关属性

### ☐ chart.grouping

用于在绘制分组柱状图时指定分组形式。可以指定的值为 grouped（在 Y 轴上的同一个单位上绘制多根柱子）或 stacked（将一根柱子分为几层颜色）。默认值为 grouped。

### ☐ chart.vmargin

用于设置每根柱子与网格线的相隔距离（以像素为单位）。默认值为 5。

### ☐ chart.strokestyle

用于设置绘制柱状图中每根柱子的边框颜色。默认值为 black。

## 11.6.2 示例程序

接下来介绍一个使用 RGraph 插件制作横向柱状图的示例程序，该示例程序的功能与代码清单 11-1 的功能大致相同，都是反映 2010 年常州第一百货公司长虹彩电销售情况，不过这一次使用横向柱状图来作为统计图。该示例程序在浏览器中打开时的显示效果如图 11-16 所示。



图 11-16 使用 RGraph 插件绘制横向柱状图

这个示例的完整代码如代码清单 11-15 所示。

代码清单 11-15 使用 RGraph 插件绘制横向柱状图

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title>使用 RGraph 插件制作横向柱状图 </title>
```



```

<script src="RGraph.common.core.js"></script>
<script src="RGraph.hbar.js"></script>
<script>
function window_onload()
{
    // 绘制横向柱状图, 指定数据
    myGraph = new RGraph.HBar('myCanvas', [1200,1300,1400,1500,3000,1900,
    2000,2100,2500,2700,1400,2600]);
    // 指定统计图标题
    myGraph.Set('chart.title','2010 年常州第一百货公司长虹彩电销售图');
    // 指定 X 轴标题
    myGraph.Set('chart.title.xaxis','销售台数');
    // 指定 Y 轴标题
    myGraph.Set('chart.title.yaxis','销售月份');
    // 指定 Y 轴的坐标轴文字
    myGraph.Set('chart.labels',['1月','2月','3月','4月','5月','6月',
    '7月','8月','9月','10月','11月','12月']);
    // 指定在坐标轴右端绘制说明销售数量的文字
    myGraph.Set('chart.labels.above', true);
    // 指定网格自动与坐标轴单位对齐
    myGraph.Set('chart.background.grid.autofit', true);
    myGraph.Set('chart.background.grid.autofit.align', true);
    // 指定标签文字所使用的空间尺寸
    myGraph.Set('chart.gutter',40);
    // 绘制横向柱状图
    myGraph.Draw();
}
</script>
</head>
<body onload="window_onload()">
<h1>使用 RGraph 插件制作横向柱状图 </h1>
<canvas id="myCanvas" width="700" height="400">
    [ 您的浏览器不支持 canvas 元素 ]
</canvas>
</body>
</html>

```

### 11.6.3 绘制分组横向柱状图

接下来介绍一个使用 RGraph 插件绘制分组横向柱状图的示例程序。该示例程序的功能与代码清单 11-3 的功能基本相同, 都是反映 2010 年常州第一百货公司长虹牌与康佳牌彩电的销售情况, 只不过这一次使用横向柱状图作为统计图。

示例的页面在浏览器中的显示效果如图 11-17 所示。



图 11-17 使用 RGraph 插件绘制分组横向柱状图

示例程序的完整代码如代码清单 11-16 所示。

代码清单 11-16 绘制分组横向柱状图

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title> 使用 RGraph 插件制作横向柱状图 </title>
<script src="RGraph.common.core.js"></script>
<script src="RGraph.Hbar.js"></script>
<script>
function window_onload()
{
    // 绘制分组横向柱状图，指定数据
    myGraph = new RGraph.HBar('myCanvas', [[1200,1600], [1300,1200],
        [1400,1200], [1500,1600], [3000,1800], [1900,1200], [2000,1600],
        [2100,1900], [2500,1100], [2700,1000], [1400,1600], [2600,1200]]);
    // 指定统计图标题
    myGraph.Set('chart.title', '2010 年常州第一百货公司彩电销售图');
    // 指定 X 轴标题
    myGraph.Set('chart.title.xaxis', '销售台数');
    // 指定 Y 轴标题
    myGraph.Set('chart.title.yaxis', '销售月份');
    // 指定柱状图图例被绘制在图例区域中
    myGraph.Set('chart.key.position', 'graph');
    // 指定图例文字
    myGraph.Set('chart.key', ['长虹', '康佳']);
    // 指定柱子颜色
    myGraph.Set('chart.colors', ['blue', 'green']);
    // 指定 Y 轴的坐标轴文字
    myGraph.Set('chart.labels', ['1月', '2月', '3月', '4月', '5月', '6月',
```

```

        '7月','8月','9月','10月','11月','12月']]);
// 指定在坐标轴右端绘制说明销售台数的文字
myGraph.Set('chart.labels.above',true);
// 指定网格自动与坐标轴单位对齐
myGraph.Set('chart.background.grid.autofit', true);
myGraph.Set('chart.background.grid.autofit.align', true);
// 指定标签文字所使用的空间尺寸
myGraph.Set('chart.gutter',45);
// 绘制分组横向柱状图
myGraph.Draw();
}
</script>
</head>
<body onload="window.onload()">
<h1> 使用 RGraph 插件制作横向柱状图 </h1>
<canvas id="myCanvas" width="900" height="400">
    [ 您的浏览器不支持 canvas 元素 ]
</canvas>
</body>
</html>

```

---

## 11.7 绘制雷达图

本节介绍如何使用 RGraph 插件绘制雷达图。在讲解示例程序前，先来看一下在绘制雷达图时所用到的属性。

### 11.7.1 绘制雷达图时所用到的属性

除了前面所讲到的绘制所有统计图时用到的属性外，绘制雷达图时将会用到的属性如下所示。

❑ chart.colors.alpha

指定值的范围为 0 到 1，用来指定雷达图中使用颜色的透明度。默认值为 null。

❑ chart.ymax

用于指定 Y 坐标轴上数字的最大值。如果不指定，则该值将被自动计算出来。默认值为 null（自动计算）。

❑ chart.background.circles

用于指定是否绘制灰色背景圆圈。默认值为 true。

❑ chart.linewidth

用于指定边框线的线宽。默认值为 1。

❑ chart.circle

用于指定雷达图中心圆圈的值。默认值为 0。

❑ chart.circle.fill

用于指定雷达图中心圆圈的填充颜色。默认值为 red。

□ chart.circle.stroke

用于指定雷达图中心圆圈的边框颜色。默认值为 black。

## 11.7.2 示例程序

接下来介绍一个绘制雷达图的示例程序。该示例程序反映了常州第一百货公司 2011 年各种彩电的销售百分比。

该示例程序在浏览器中的运行结果如图 11-18 所示。

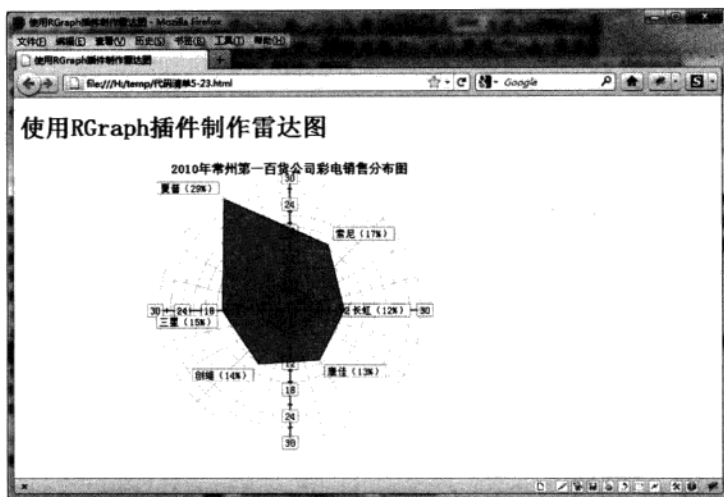


图 11-18 使用 RGraph 插件制作雷达图

该示例的完整代码如代码清单 11-17 所示。

代码清单 11-17 使用 RGraph 插件制作雷达图

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title> 使用 RGraph 插件制作雷达图 </title>
<script src="RGraph.common.core.js"></script>
<script src="RGraph.tradar.js"></script>
<script>
function window_onload()
{
    // 绘制雷达图，获取数据
    var tradar=new RGraph.Tradar('myCanvas',[12,13,14,15,29,17]);
    // 指定雷达图标题
    tradar.Set('chart.title', '2010 年常州第一百货公司彩电销售分布图');
    // 指定标签文字
```

```

    radar.Set('chart.labels', ['长虹 (12%)', '康佳 (13%)', '创维 (14%)',
    '三星 (15%)', '夏普 (29%)', '索尼 (17%)']);
    // 指定绘制背景灰色圆
    radar.Set('chart.background.circles', true);
    // 指定雷达图颜色
    radar.Set('chart.color', 'rgba(255,0,0,0.5)');
    // 指定中央圆直径为 5
    radar.Set('chart.circle', 5);
    // 指定中央圆圈的填充颜色
    radar.Set('chart.circle.fill', 'rgba(200,255,200,0.5)');
    // 绘制雷达图
    radar.Draw();
}
</script>
</head>
<body onload="window.onload()">
<h1>使用 RGraph 插件制作雷达图 </h1>
<canvas id="myCanvas" width="700" height="400">
    [您的浏览器不支持 canvas 元素]
</canvas>
</body>
</html>

```

---

## 11.8 增强用户体验

通过使用 RGraph 插件中的一些属性，可以增强用户体验，使用户可以对所绘制的统计图进行简单操作。

### 11.8.1 通过拖曳来缩放统计图的尺寸

通过将统计图的公共属性 `chart.resizable` 的属性值设为 `true`，使用户通过拖曳操作来自定义统计图的尺寸。使用该属性的步骤有两步：

- 1) 引用 `RGraph.common.resizing.js` 脚本文件，代码如下所示。

```
<script src="RGraph.common.resizing.js"></script>
```

- 2) 在统计图的属性设置中将统计图的 `chart.resizable` 的属性值设为 `true`，代码如下所示。

```
myGraph.Set('chart.resizable', true);
```

完成了这两步操作后，统计图右下角将会出现一个十字拖放符号，用户可以通过拖曳该符号来放大或缩小统计图的尺寸，如图 11-19 所示。

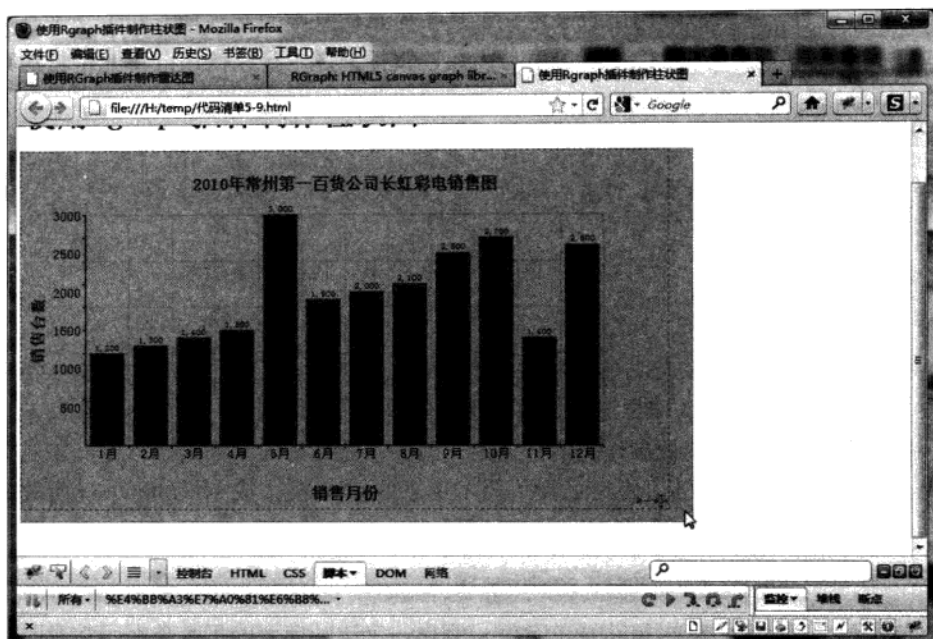


图 11-19 用户通过 chart.resizable 属性自定义统计图尺寸

## 11.8.2 制作工具条提示信息

### 1. 概述

利用 RGraph 插件可以为统计图制作各种丰富多彩的工具条提示信息。

从本质上来说, RGraph 插件通过一个 div 元素来显示工具条提示信息。因此, 可以使用 HTML 语言来制作工具条提示信息, 以显示图像 (使用 img 元素) 和视频 (使用 video 元素) 等信息。

### 2. 工具条提示信息的指定方法

使用 chart.tooltips 属性为统计图添加工具条提示信息。具体步骤分为如下两步:

1) 引用 RGraph.common.tooltips.js 脚本文件, 代码如下所示。

```
<script src="RGraph.common.tooltips.js"></script>
```

2) 使用 chart.tooltips 属性定义工具条提示信息, 代码如下所示。

```
var pie=new RGraph.Pie('myCanvas',[12000,13000,14000,15000,30000,19000]);
pie.Set('chart.tooltips',[ '长虹 (12%) ', '康佳 (13%) ', '创维 (14%) ',
'三星 (15%) ', '夏普 (29%) ', '索尼 (17%) ']);
```

从本质上来说, 必须利用字符串来定义工具条提示信息中的每一项 (可以是 HTML 语言), 也可以通过函数来进行定义, 但是这些函数必须返回字符串。可以使用函数来定义整

个工具条提示信息，也可以使用函数来定义工具条提示信息中的某一项。在定义工具条提示信息中的每一项时，可以同时使用字符串与函数。概括来说，可以使用以下几种方法定义工具条提示信息。

□ 使用字符串数组，代码如下所示。

```
pie.Set('chart.tooltips', ['长虹 (12%)', '康佳 (13%)', '创维 (14%)',  
'三星 (15%)', '夏普 (29%)', '索尼 (17%)']);
```

□ 使用函数数组，每一个函数必须返回一个字符串，代码如下所示。

```
myGraph.Set('chart.tooltips', [getTooltip1, getTooltip2, getTooltip3]);
```

□ 使用一个函数，该函数必须返回一个字符串（如果只有一个工具条项）或一个字符串数组（每一个工具条项使用字符串数组中对应序号的字符串）。该函数在显示工具条提示信息的 div 元素被创建之前被调用，因此不能被访问。如果想自定义工具条提示信息的外观样式，必须使用工具条提示信息的 CSS 类，或 setTimeout() 函数。使用函数定义工具条提示信息的代码如下所示。

```
myGraph.Set('chart.tooltips', getTooltip);
```

□ 使用一个由 div 元素的 id 构成的数组。制作一个比较庞大的工具条提示信息时可以使用这种方法。这些 div 元素的内部 HTML 语言（innerHTML）将作为工具条提示信息中的每一项来利用。注意，只有 div 元素的内容被利用，div 元素本身不被利用。因此可以通过将这些 div 元素的 display 这个 CSS 属性的属性值设为 none 来隐藏工具条提示信息。使用由 div 元素的 id 构成的数组来定义工具条提示信息的代码如下所示。

```
myGraph.Set('chart.tooltips',  
['id:myDiv1', 'id:myDiv2', 'id:myDiv3', 'id:myDiv4', 'id:myDiv5']);
```

### 3. 在工具条提示信息中使用统计图

使用 RGraph 插件可以在工具条提示信息中显示统计图。通过使用特定事件可以很容易实现这一操作，只需在 ontooltip 事件中使用创建统计图的函数即可。可以在该函数内书写工具条提示信息中所显示的统计图的 HTML 代码。具体来说，分为以下几个步骤：

- 1) 为工具条提示信息书写 HTML 代码（在代码中使用 canvas 元素）。
- 2) 在 RGraph 的 ontooltip 事件中书写显示工具条信息提示时调用的函数。
- 3) 在该函数中书写工具条提示信息中显示的统计图代码。

使用 RGraph.Registry.Get('chart.tooltip') 方法可以获得工具条提示信息所使用的 div 元素，使用 RGraph.Registry.Get('chart.tooltip').\_\_index\_\_ 属性可以获得每一个工具条提示信息的编号。

接下来看一个示例程序，该示例程序使用柱状图来显示常州第一百货公司 2010 年每个月彩电的销售情况，在页面中单击柱状图中每根柱子将弹出工具条提示信息，在工具条提示信息中使用饼图来显示该月中各种彩电的销售数量的分布情况。

该示例程序在浏览器中打开时的页面显示效果如图 11-20 所示。

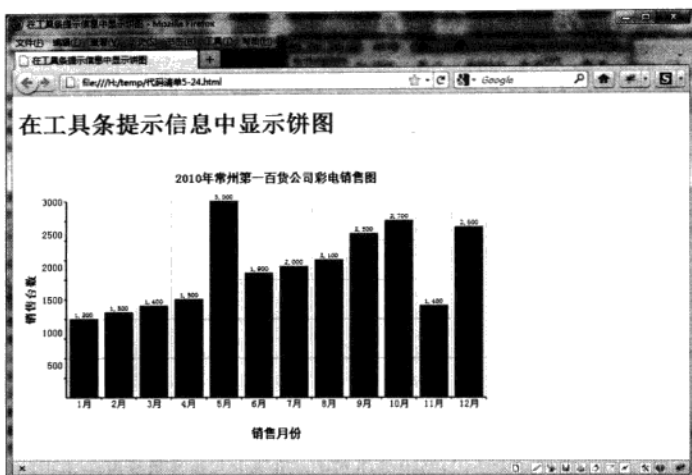


图 11-20 示例页面在浏览器中打开时的显示效果

用户单击每根柱子时将出现工具条提示信息，在工具条提示信息中显示该月各种彩电的销售数量的分布情况，如图 11-21 所示。

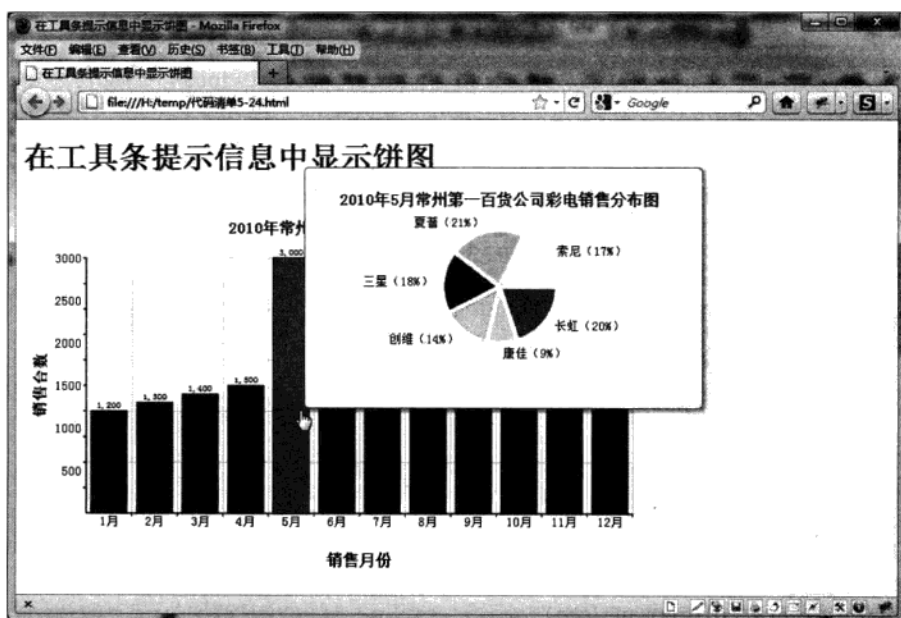


图 11-21 用户单击柱子时将在工具条提示信息中显示饼图

该示例的完整代码如代码清单 11-18 所示。



## 代码清单 11-18 在工具条提示信息中显示饼图

```

<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title> 在工具条提示信息中显示饼图 </title>
<script src="RGraph.common.core.js"></script>
<script src="RGraph.common.tooltips.js" ></script>
<script src="RGraph.bar.js"></script>
<script src="RGraph.pie.js"></script>
</script>

function window_onload()
{
    // 绘制柱状图, 指定数据
    myGraph = new RGraph.Bar('myCanvas', [1200,1300,1400,1500,3000,1900,2000,
    2100,2500,2700,1400,2600]);
    // 指定统计图标题
    myGraph.Set('chart.title', '2010 年常州第一百货公司彩电销售图 ');
    // 指定 X 轴标题
    myGraph.Set('chart.title.xaxis', '销售月份 ');
    // 指定 Y 轴标题
    myGraph.Set('chart.title.yaxis', '销售台数 ');
    // 指定 X 轴的坐标轴文字
    myGraph.Set('chart.labels', ['1 月 ', '2 月 ', '3 月 ', '4 月 ', '5 月 ', '6 月 ',
    '7 月 ', '8 月 ', '9 月 ', '10 月 ', '11 月 ', '12 月 ']);
    // 指定 Y 轴的坐标轴文字
    myGraph.Set('chart.ylabels.specific', ['3000', '2500', '2000', '1500',
    '1000', '500']);
    // 指定在坐标轴顶部绘制的说明销售数量的文字
    myGraph.Set('chart.labels.above', true);
    // 指定网格自动与坐标轴单位对齐
    myGraph.Set('chart.background.grid.autofit', true);
    myGraph.Set('chart.background.grid.autofit.align', true);
    // 指定标签文字所使用的空间尺寸
    myGraph.Set('chart.gutter', 65);
    // 指定工具条提示信息
    myGraph.Set('chart.tooltips', function (idx) {
        return '<canvas id="__tooltip_canvas__"
        width="400" height="250">[ 您的浏览器不支持 canvas 元素 ]</canvas>';});
    // 绘制柱状图
    myGraph.Draw();
    // 添加显示工具条提示信息时的事件处理
    RGraph.AddCustomEventListener(myGraph, 'ontooltip', CreateTooltipGraph);
}
// 绘制饼图
function CreateTooltipGraph(obj)
{
    // 显示月份时使用的数组
    var months=['1 月 ', '2 月 ', '3 月 ', '4 月 ', '5 月 ', '6 月 ', '7 月 ', '8 月 ', '9 月 ',

```

```

'10月','11月','12月'];
// 各种彩电 12 个月的销售数量, 仅供示例用
var datas = [
[1200,1300,1400,1500,3000,1900],
[790,990,670,760,770,1100],
[1200,560,700,990,670,1100],
[780,1900,950,700,760,780],
[860,380,590,780,930,780],
[880,1100,780,680,960,1300],
[530,780,940,570,790,1200],
[1300,600,630,710,720,1100],
[920,850,750,820,480,1100],
[780,390,600,500,680,1200],
[760,1000,730,1000,630,880],
[480,1000,1200,1000,650,440]
];
// 各种彩电占总销售数量的百分比
var labels=[
['长虹 (12%)', '康佳 (13%)', '创维 (14%)', '三星 (15%)', '夏普 (29%)',
'索尼 (17%)'],
['长虹 (16%)', '康佳 (19%)', '创维 (13%)', '三星 (15%)', '夏普 (15%)',
'索尼 (22%)'],
['长虹 (23%)', '康佳 (11%)', '创维 (13%)', '三星 (19%)', '夏普 (13%)',
'索尼 (22%)'],
['长虹 (13%)', '康佳 (32%)', '创维 (16%)', '三星 (12%)', '夏普 (13%)',
'索尼 (14%)'],
['长虹 (20%)', '康佳 (9%)', '创维 (14%)', '三星 (18%)', '夏普 (21%)',
'索尼 (17%)'],
['长虹 (15%)', '康佳 (19%)', '创维 (14%)', '三星 (12%)', '夏普 (17%)',
'索尼 (23%)'],
['长虹 (11%)', '康佳 (16%)', '创维 (20%)', '三星 (12%)', '夏普 (16%)',
'索尼 (25%)'],
['长虹 (26%)', '康佳 (12%)', '创维 (12%)', '三星 (14%)', '夏普 (14%)',
'索尼 (22%)'],
['长虹 (19%)', '康佳 (17%)', '创维 (15%)', '三星 (17%)', '夏普 (10%)',
'索尼 (22%)'],
['长虹 (19%)', '康佳 (9%)', '创维 (14%)', '三星 (12%)', '夏普 (16%)',
'索尼 (30%)'],
['长虹 (15%)', '康佳 (20%)', '创维 (15%)', '三星 (20%)', '夏普 (13%)',
'索尼 (17%)'],
['长虹 (10%)', '康佳 (21%)', '创维 (25%)', '三星 (21%)', '夏普 (14%)',
'索尼 (9%)']
]
// 获取触发事件的工具条提示信息项的编号
var idx = RGraph.Registry.Get('chart.tooltip').__index__;
// 绘制饼图, 指定数据
var pie=new RGraph.Pie('__tooltip_canvas__',datas[idx]);
// 绘制饼图标题
pie.Set('chart.title',
'2010 年 '+months[idx]+' 常州第一百货公司彩电销售分布图');

```

```

    // 指定标签文字
    pie.Set('chart.labels', labels[idx]);
    // 指定饼块分隔线宽
    pie.Set('chart.linewidth', 5);
    // 指定饼块分隔线颜色
    pie.Set('chart.strokestyle', 'white');
    // 指定标签文字所使用的空间尺寸
    pie.Set('chart.gutter', 65);
    // 绘制饼图
    pie.Draw();
}

</script>
</head>
<body onload="window.onload()">
<h1> 在工具条提示信息中显示饼图 </h1>
<canvas id="myCanvas" width="700" height="400">
    [ 您的浏览器不支持 canvas 元素 ]
</canvas>
</body>
</html>

```

#### 4. 工具条提示信息的样式及显示特效

可以使用工具条提示信息的默认样式类 `Rgraph_tooltip`，也可以使用 `chart.tooltips.css` 的 `class` 属性来自定义工具条提示信息的样式类，代码如下所示。

```

bar.Set('chart.tooltips.css.class', 'bar_chart_tooltips_css');
<style>
.bar_chart_tooltips_css{
    background-color: white ! important;
    border: 2px solid black ! important;
    padding: 3px;
}
</style>

```

可以使用 `chart.tooltips.effect` 属性来指定工具条信息提示在显示时使用的特效，可以指定的值有以下几种。

- `fade`: 淡入
- `expand`: 从工具条提示信息中央向外扩展
- `contract`: 从外部向工具条提示信息中央缩进
- `none`: 不使用特效

#### 5. 重载工具条提示信息的默认行为

可以通过对 `chart.tooltips.override` 属性指定一个函数来重载显示工具条提示信息时的默认行为。该重载函数必须使用以下几个参数。

- canvas: 绘制统计图时所用的 canvas 元素
  - text: 工具条提示信息项中所显示的文字
  - x: 工具条提示信息项在整个页面中 X 轴上的坐标点
  - y: 工具条提示信息项在整个页面中 Y 轴上的坐标点
  - index: 工具条提示信息项在整个工具条提示信息中的序号
- 重载函数的使用方法如下所示。

```
<script>
    function tooltip_override (canvas, text, x, y, idx)
    {
        alert('In tooltip override function...');
    }
    myObj.Set('chart.tooltips.override', tooltip_override);
</script>
```

### 11.8.3 制作上下文菜单

通过 RGraph 插件可以为制作出来的统计图添加上下文菜单功能。所谓上下文菜单，是指用户在统计图上单击右键时弹出的菜单。可以定制上下文菜单中的每一个菜单项。通过上下文菜单的制作，可以很容易地为统计图添加各种功能。

注意，在 Opera 浏览器中，并不是通过单击右键来弹出上下文菜单，而是通过双击左键的方式来弹出上下文菜单。

可以通过使用 CSS 样式类的方法来定制上下文菜单的样式，在指定上下文菜单的背景所使用的样式时需要将类名指定为“RGraph\_contextmenu\_background”，在指定上下文菜单所使用的样式时需要将类名指定为“RGraph\_contextmenu”，在指定菜单项所使用的样式时需要将类名指定为“RGraph\_contextmenu\_item”，指定方法如下所示。

```
<style type="text/css">
.RGraph_contextmenu {
}
.RGraph_contextmenu_item {
}
.RGraph_contextmenu_background {
}
</style>
```

使用类似如下所示的方法来指定上下文菜单中的菜单项。

```
myBar.Set('chart.contextmenu', [['菜单项 1', Callback1],
                                ['菜单项 2', Callback2]
]);
```

如上所示，使用一个二维数组来指定菜单项，在二维数组的第一维数组中使用字符串来指定菜单项上的显示文字，在第二维数组中指定一个回调函数，在该回调函数中指定用户单

击该菜单项后要执行的处理。

可以在上下文菜单项之间指定一个分隔线，方法是将该菜单项指定为 `null`，代码如下所示。

```
myBar.Set('chart.contextmenu', [
    ['菜单项 1', Callback1],
    null,
    ['菜单项 2', Callback2]
]);
```

在指定菜单项的时候，可以指定具有子菜单的菜单项，方法如下所示。

```
bar.Set('chart.contextmenu',
[['放大统计图', RGraph.Zoom],
 ['应用', [['登录 ...', function () {ModalDialog.Show('modaldialog_login',
300);}]] ],
null,
['取消', function () {}]
]);
```

使用上下文菜单的示例程序可以参考代码清单 11-5，在该示例程序中使用了一个上下文菜单，在该上下文菜单中指定了两个菜单项，分别为“将一根柱子分为几层颜色”与“在一个坐标单位上绘制多根柱子”。用户单击“将一根柱子分为几层颜色”菜单项时使用在一个坐标单位上只绘制一根柱子，将每根柱子分为几层颜色的方法来绘制分组柱状图，用户单击“在一个坐标单位上绘制多根柱子”菜单项时使用在一个坐标单位上绘制多根柱子的方法来绘制分组柱状图。

## 11.8.4 放大统计图

可以对利用 RGraph 插件制作出来的统计图进行放大。放大统计图时所使用的属性在 11.2 节的“统计图放大相关属性”中已经介绍，这里不再赘述。

可以使用以下两种方式将统计图进行放大。

### 1. canvas 模式

首先介绍一个使用 `canvas` 模式将统计图进行放大的示例程。

该示例程序在代码清单 11-1 的基础上添加了使用统计图放大功能。当用户在统计图上单击右键时将出现一个上下文菜单，在该菜单项中指定了一个菜单项“放大统计图”，如图 11-22 所示。

用户单击“放大柱状图”菜单项后，将弹出一个图片，在该图片中显示放大的柱状图，如图 11-23 所示。

用户单击该放大图片之外的区域时，该放大图片将被关闭。

示例程序的完整代码如代码清单 11-19 所示。注意，在该示例代码中使用了 `chart.contextmenu` 属性来指定上下文菜单，在菜单项中指定了“放大柱状图”菜单项，并且指定

在用户单击该菜单项时使用 RGraph.Zoom 方法来显示放大的统计图。

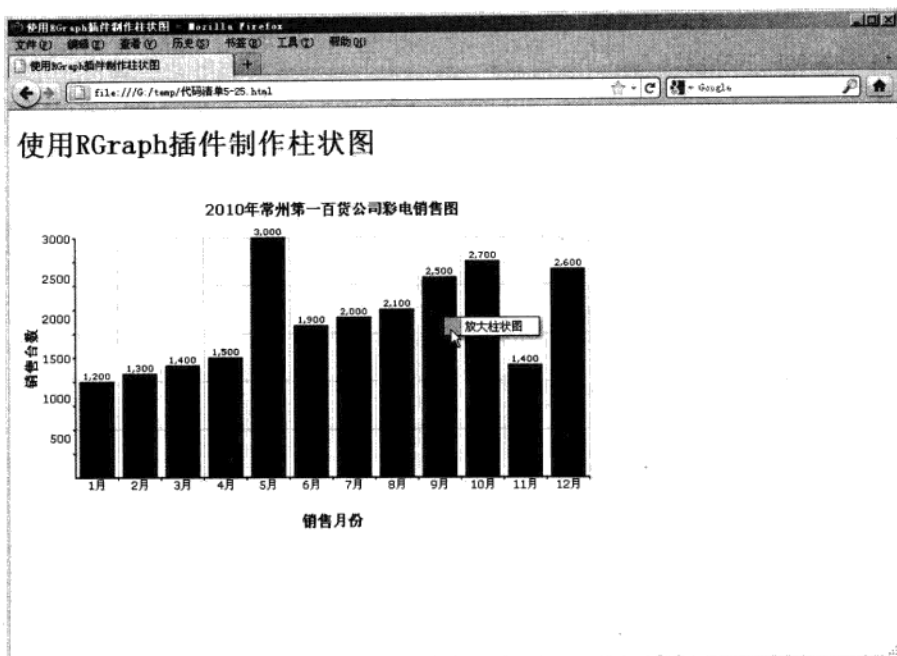


图 11-22 单击右键出现上下文菜单

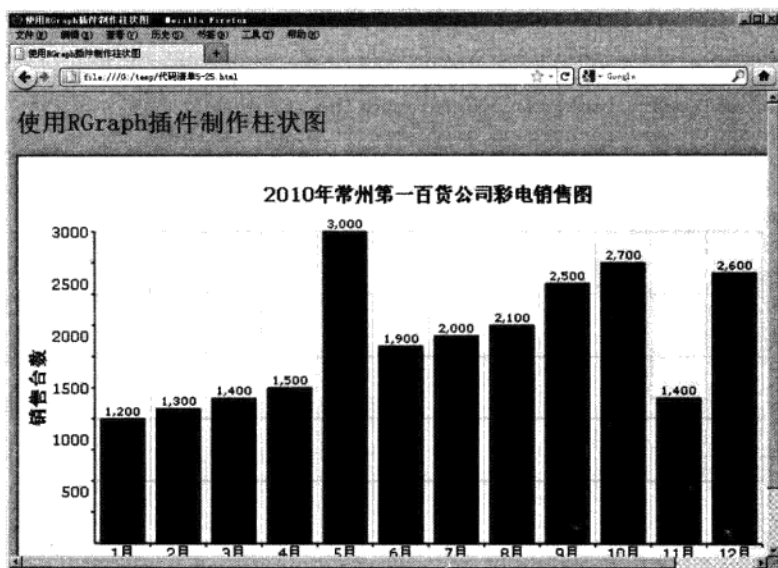


图 11-23 用户单击“放大柱状图”菜单项后弹出放大的柱状图

---

```

<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title> 使用 RGraph 插件制作柱状图 </title>
<script src="RGraph.common.core.js"></script>
<script src="RGraph.common.context.js"></script>
<script src="RGraph.common.zoom.js"></script>
<script src="RGraph.bar.js"></script>
<script>
function window_onload()
{
    // 绘制柱状图, 指定数据
    myGraph = new RGraph.Bar('myCanvas', [1200,1300,1400,1500,3000,1900,2000,
    2100,2500,2700,1400,2600]);
    // 指定统计图标题
    myGraph.Set('chart.title','2010 年常州第一百货公司彩电销售图 ');
    // 指定 X 轴标题
    myGraph.Set('chart.title.xaxis',' 销售月份 ');
    // 指定 Y 轴标题
    myGraph.Set('chart.title.yaxis',' 销售台数 ');
    // 指定 X 轴的坐标轴文字
    myGraph.Set('chart.labels',[ '1 月 ', '2 月 ', '3 月 ', '4 月 ', '5 月 ', '6 月 ',
    '7 月 ', '8 月 ', '9 月 ', '10 月 ', '11 月 ', '12 月 ']);
    // 指定 Y 轴的坐标轴文字
    myGraph.Set('chart.ylabels.specific',['3000','2500','2000','1500',
    '1000','500']);
    // 指定在坐标轴顶部绘制说明销售数量的文字
    myGraph.Set('chart.labels.above', true);
    // 指定网格自动与坐标轴单位对齐
    myGraph.Set('chart.background.grid.autofit', true);
    myGraph.Set('chart.background.grid.autofit.align', true);
    // 指定标签文字所使用的空间尺寸
    myGraph.Set('chart.gutter',65);
    // 指定弹出菜单, 用户单击菜单项时放大统计图
    myGraph.Set('chart.contextmenu',[[' 放大柱状图 ', RGraph.Zoom]]);
    // 绘制柱状图
    myGraph.Draw();
}

</script>
</head>
<body onload="window_onload()">
<h1> 使用 RGraph 插件制作柱状图 </h1>
<canvas id="myCanvas" width="700" height="400">
    [ 您的浏览器不支持 canvas 元素 ]
</canvas>
</body>
</html>

```

---

## 2. 缩略图模式

第二种放大统计图的方法被称为缩略图方法，这种方法不是针对整个统计图进行的，而是通过一个“放大镜”来放大统计图中的局部。当光标移动到统计图上时会出现一个“放大镜”，在该“放大镜”中放大显示鼠标指针所在的局部区域，随着鼠标指针在统计图上的移动来更新“放大镜”中的放大区域。

接下来在代码清单 11-20 的基础上进行修改，使其能够同时使用两种统计图的放大方法。当鼠标指针移动到统计图上时会出现一个“放大镜”，将鼠标指针所在局部区域放大显示，并且随着鼠标指针在统计图上的移动来更新“放大镜”中的放大区域，如图 11-24 所示。

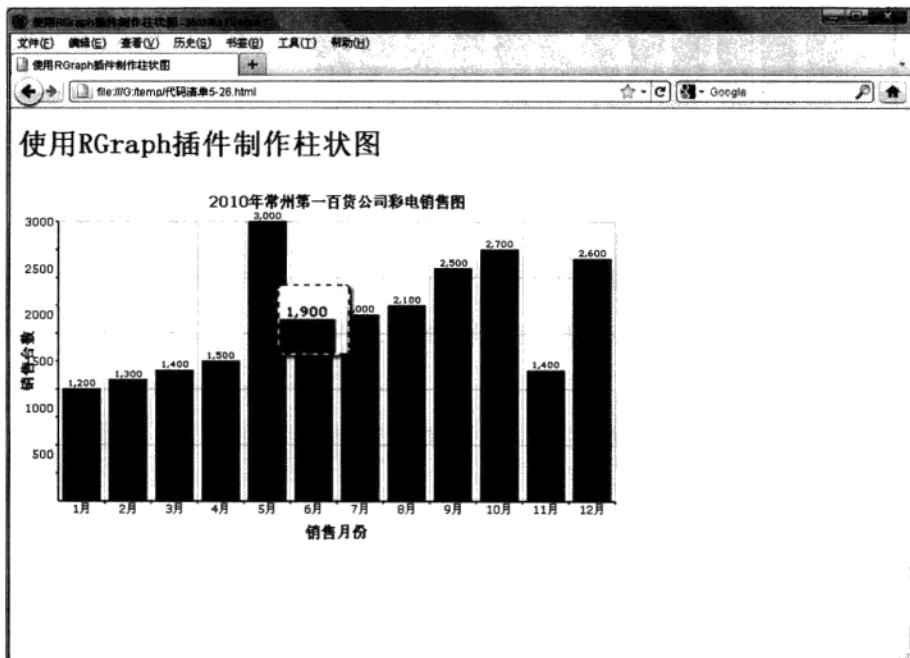


图 11-24 使用缩略图方式放大统计图

另外，通过如下所示的样式可以使“放大镜”由方形变为圆形，使其看起来更加具有放大镜的显示效果。

```
<style>
.RGraph_zoom_window {
    border-radius: 50px ! important;
}
</style>
```

使用该样式后，放大统计图时的显示效果如图 11-25 所示。



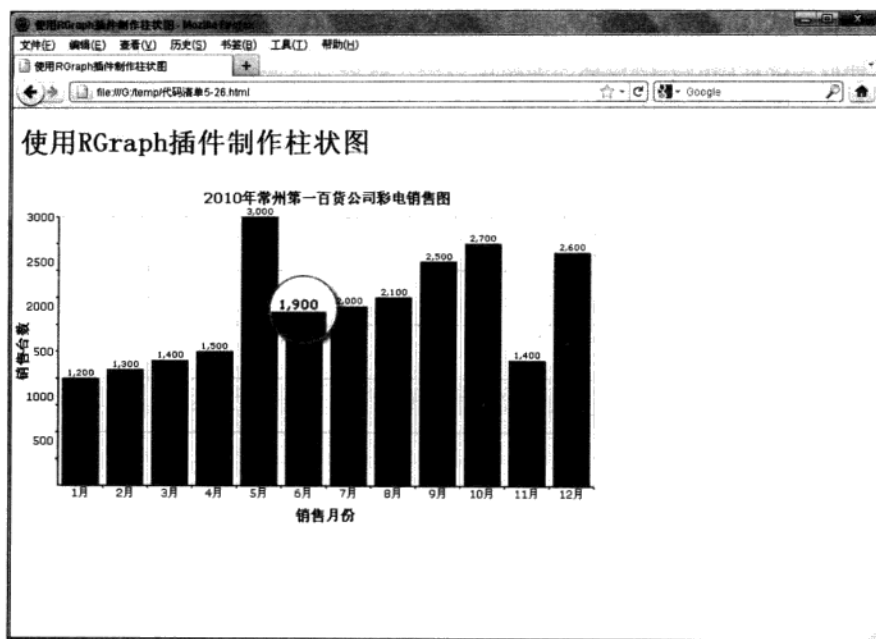


图 11-25 使用圆形“放大镜”的放大效果

这个示例程序的完整代码如代码清单 11-20 所示，注意这段代码中有关放大统计图的几个属性的使用方法。

代码清单 11-20 使用缩略图模式放大统计图

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title> 使用 RGraph 插件制作柱状图 </title>
<script src="RGraph.common.core.js"></script>
<script src="RGraph.common.context.js"></script>
<script src="RGraph.common.zoom.js"></script>
<script src="RGraph.bar.js"></script>
</script>
function window_onload()
{
    // 绘制柱状图，指定数据
    myGraph = new RGraph.Bar('myCanvas', [1200,1300,1400,1500,3000,1900,2000,
    2100,2500,2700,1400,2600]);
    // 指定统计图标题
    myGraph.Set('chart.title','2010 年常州第一百货公司彩电销售图');
    // 指定 X 轴标题
    myGraph.Set('chart.title.xaxis','销售月份');
    // 指定 Y 轴标题
    myGraph.Set('chart.title.yaxis','销售台数');
    // 指定 X 轴的坐标轴文字
```

```

myGraph.Set('chart.labels', ['1月', '2月', '3月', '4月', '5月', '6月',
    '7月', '8月', '9月', '10月', '11月', '12月']);
// 指定 Y 轴的坐标轴文字
myGraph.Set('chart.ylabels.specific', ['3000', '2500', '2000', '1500',
    '1000', '500']);
// 指定在坐标轴顶部绘制说明销售数量的文字
myGraph.Set('chart.labels.above', true);
// 指定网格自动与坐标轴单位对齐
myGraph.Set('chart.background.grid.autofit', true);
myGraph.Set('chart.background.grid.autofit.align', true);
// 指定标签文字所使用的空间尺寸
myGraph.Set('chart.gutter', 65);
// 使用缩略图模式放大统计图
myGraph.Set('chart.zoom.mode', 'thumbnail');
// 放大镜的宽度
myGraph.Set('chart.zoom.thumbnail.width', 100);
// 放大镜的高度
myGraph.Set('chart.zoom.thumbnail.height', 100);
// 指定弹出菜单, 用户单击菜单项时放大统计图
myGraph.Set('chart.contextmenu', [['放大柱状图', RGraph.Zoom]]);
// 绘制柱状图
myGraph.Draw();
}

</script>
<style>
    .RGraph_zoom_window {
        border-radius: 50px ! important;
    }
</style>
</head>
<body onload="window.onload()">
<h1>使用 RGraph 插件制作柱状图 </h1>
<canvas id="myCanvas" width="700" height="400">
    [ 您的浏览器不支持 canvas 元素 ]
</canvas>
</body>
</html>

```

### 11.8.5 允许用户注解统计图

如果使用 RGraph 插件来进行统计图的绘制, 可以让用户自己在统计图上添加注解, 这样, 当用户保存统计图的时候, 可以将这个注解一起保存, 在一些支持本地存储的浏览器 (例如 Chrome10 以上, Opera10 以上浏览器) 中, 当用户再次打开这个统计图页面的时候, 该注解也会一起被显示。

以上处理需要通过以下两个步骤来实现:

1) 添加 RGraph.common.annotate.js 脚本文件的引用, 代码如下所示。

```
<script src="RGraph.common.annotate.js"></script>
```

2) 将 chart.annotatable 属性的属性值设为 true, 代码如下所示。

```
myGraph.Set('chart.annotatable', true);
```

在代码清单 11-20 的示例代码中完成这两个步骤后, 用户就可以自行对柱状图进行注解, 如图 11-26 所示。

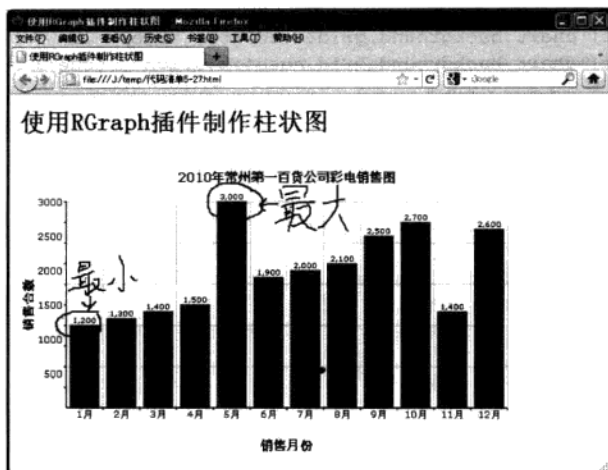


图 11-26 用户可以自行在统计图上添加注解

可以使用如下方法在上下文菜单中添加一个“显示调色盘”菜单项, 用户单击该菜单项后页面上将显示一个调色盘, 用户可以使用这个调色盘来选择注解所使用的颜色。

```
myGraph.Set('chart.contextmenu',
[
    ['显示调色盘', RGraph.Showpalette],
]);
```

用户在柱状图上单击右键, 将出现一个上下文菜单, 该菜单中具有“显示调色盘”菜单项, 如图 11-27 所示。

用户单击“显示调色盘”菜单项后, 页面上将会出现一个调色盘, 如图 11-28 所示。

用户可以在调色盘中选择颜色, 然后使用该颜色来进行注解, 如图 11-29 所示。

最后, 可以使用如下代码在上下文菜单中添加“擦除注解”菜单项, 用户单击该菜单项后可以擦除自己不满意的注解。

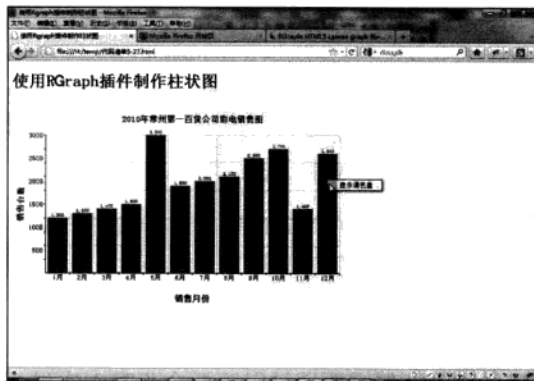


图 11-27 单击右键后出现上下文菜单

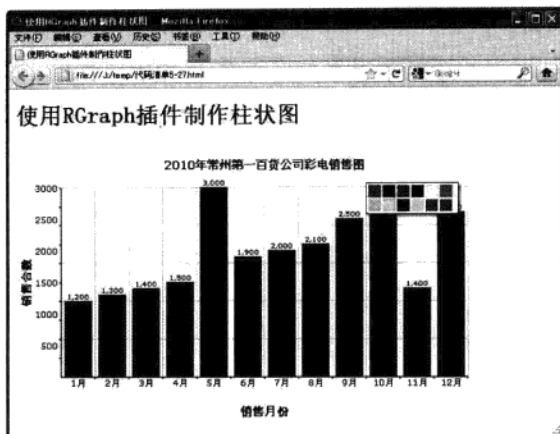


图 11-28 用户单击“显示调色盘”菜单项后页面上出现一个调色盘

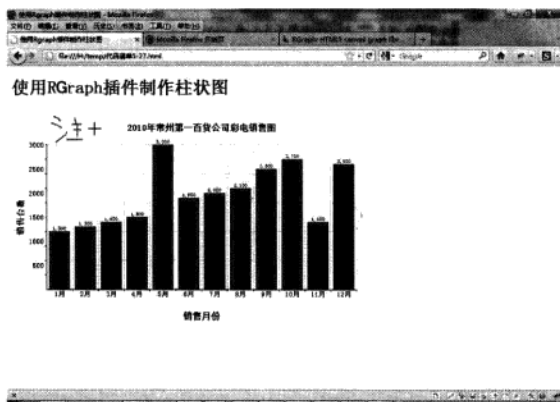


图 11-29 用户可以使用选择的颜色进行注解

```
myGraph.Set('chart.contextmenu',
[
  [' 擦除注解 ', function () {RGraph.Clear(myGraph.canvas);myGraph.Draw();}]
]);
```

## 11.9 本章小结

本章对一个以 HTML 5 中的 canvas 元素及 Canvas API 为基础制作出来的，专门用于绘制各种统计图的插件——RGraph 统计图绘制插件进行了一个比较全面的、系统的介绍。希望通过对本章的阅读，读者能够对于这个插件有一个比较细致的了解，能够在自己的 Web 应用程序中灵活运用这个插件，制作出各种复杂的，功能强大的统计图。

下一章将对 HTML 5 中专门用来绘制三维图形、图像以及动画的 API——WebGL API 进行基本介绍。



## 第 12 章

# 使用 WebGL 开发三维图形图像

### 本章内容

- ☐ WebGL 概述
- ☐ 使用 WebGL 绘制三角形与矩形
- ☐ 使用颜色绘制彩色三角形与矩形
- ☐ 制作三维动画
- ☐ 制作三维物体
- ☐ 使用纹理
- ☐ 键盘输入与纹理过滤
- ☐ 本章小结

本章结合一些示例来向读者介绍 WebGL 的基础知识，目前各浏览器对 WebGL 的支持情况，以及如何使用 WebGL 来开发 3D 图形、图像与动画。

首先需要说明的是，阅读本章内容的前提是读者要有比较丰富的编程经验，但并不要求读者具有 3D 图形开发的经验。本章的目标是使读者读懂本章示例代码中每一句代码的含义，以便能够在 Web 网页上进行 3D 图形、图像与动画的开发。

## 12.1 WebGL 概述

### 12.1.1 WebGL 的基础知识

WebGL 是一种 3D 绘图标准，它的全名为“Web Graphics Library”，是由 OpenGL 的管理组织 Khronos Group 基于 OpenGL ES 2.0 所制定的跨平台的 Web 专用的一种 3D 绘图标准。目前除了 Microsoft Internet Explorer 之外的几家主要的浏览器公司，都是 WebGL 标准工作组的成员，其中包括 Apple 公司的 Safari 浏览器、Google 公司的 Chrome 浏览器、Mozilla 公司的 Firefox 浏览器与 Opera 公司的 Opera 浏览器。

使用 WebGL 的好处在于：如果浏览器支持 WebGL，就可以在不需要额外安装插件（plug-in）的情况下，使用 JavaScript 和 OpenGL ES 2.0 语言开发出具有 3D 显示效果的图形、图像、动画甚至 3D 游戏，同时也可以通过显卡来实现 3D 图形、图像或动画在浏览器中显示时的硬件加速功能。在网页中显示时，WebGL 总是把最终渲染结果显示在 HTML 5 的 canvas 元素中。由于 WebGL 使用 canvas 元素来进行显示，所以它基本上也被容纳在整个 DOM（Document Object Model）架构中。

要使用 WebGL 来进行 3D 场景和 3D 动画的开发，开发者首先必须具备 HTML、JavaScript 以及 OpenGL ES 2.0 的基础知识。

### 12.1.2 进行 WebGL 开发之前的准备工作

要想使用 WebGL 来开发 3D 场景和 3D 动画，首先要做好以下准备工作：

- 1) 使用 Windows 的用户要确保已经安装了 Microsoft DirectX runtime。访问微软的官方网站，可以很容易地下载 Microsoft DirectX runtime。
- 2) 确保已经下载并安装了显卡的最新驱动程序。
- 3) 在浏览器方面，可以选择 Firefox 浏览器或 Chrome 浏览器。目前（到 2011 年 5 月 1 日为止）最新的 Firefox 浏览器（4.0.1 版本）与 Chrome 浏览器（Chrome 10）都对 WebGL 提供了支持。可以通过浏览器访问“<http://webglreport.sourceforge.net/>”这个网址来查看浏览器是否支持 WebGL。例如，笔者使用最新的 Firefox 浏览器与最新的 Chrome 浏览器访问这个网址，检测报告中显示这两款最新的浏览器都对 WebGL 提供了支持。使用 Firefox 浏览器访问时显示结果如图 12-1 所示。

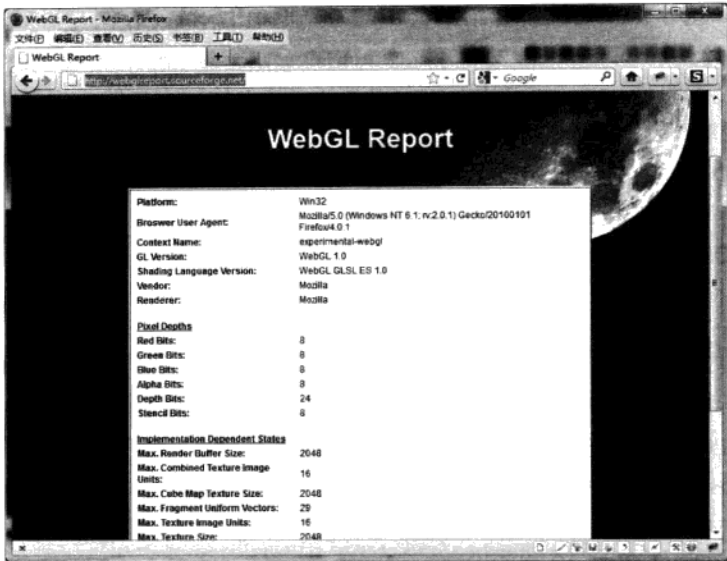


图 12-1 WebGL 报告显示最新 Firefox 浏览器支持 WebGL

使用 Chrome 浏览器访问时显示结果如图 12-2 所示。



图 12-2 WebGL 报告显示最新 Chrome 浏览器支持 WebGL

在不支持 WebGL 的浏览器（例如 Internet Explorer 浏览器）中，WebGL 报告的显示结果如图 12-3 所示。



图 12-3 WebGL 报告显示 Internet Explorer 浏览器不支持 WebGL

注意，最新版 Firefox 浏览器对于 WebGL 的支持情况也要由所使用的显卡来决定。在使用了某些显卡的机器上，最新版的 Firefox 浏览器不支持 WebGL，在这种情况下，请改用最新版的 Chrome 浏览器。

另外，对 Opera 浏览器来说，现在已经有一个预览版支持 WebGL，读者可以通过“<http://my.opera.com/core/blog/2011/02/28/webgl-and-hardware-acceleration-2>”这个网址来下载该版本的浏览器。该浏览器对于 WebGL 所提供的支持的稳定性比不上 Firefox 浏览器与 Chrome 浏览器，同时它要求显卡支持 OpenGL 2.0。

对 Safari 浏览器来说，目前只有运行在 Snow Leopard (OS X 10.6) 操作系统的苹果机上的 Safari 浏览器可以支持 WebGL。

对 Snow Leopard (OS X 10.5) 操作系统或 Linux 操作系统来说，支持 WebGL 的有 Firefox 浏览器和 Chromium 浏览器（基于 Chrome 浏览器的一种开源浏览器）。

如果使用的是 Snow Leopard 操作系统，需要执行以下步骤：

- 1) 确保拥有最新版本的 Safari 操作系统。
- 2) 从“<http://nightly.webkit.org/>”这个网址上下载“WebKit nightly build”。
- 3) 打开终端，运行如下命令。

```
defaults write com.apple.Safari WebKitWebGLEnabled -bool YES
```

- 4) 运行刚刚安装的 WebKit 应用程序。



## 12.2 使用 WebGL 绘制三角形与矩形

本节从在网页上分别绘制一个具有 3D 效果的三角形与一个具有 3D 效果的矩形着手,向读者介绍一些使用 WebGL 进行绘图的基础知识。正确理解本节中的内容,理解 WebGL 绘图知识的其他部分就更容易了。

### 12.2.1 下载并使用脚本文件

在学习本章内容之前,要先下载 glMatrix-0.9.5.min.js 脚本文件,可以从“<https://github.com/gpjt/webgl-lessons>”这个网址上下载该脚本文件(本章中其他利用 WebGL 进行 3D 图形、图像和动画制作所需要的脚本文件也可以从这个网址上下载)。该脚本文件的主要功能是用来处理矩阵操作与 vector 动态数组操作。

### 12.2.2 页面显示效果

接下来看一下本示例程序在浏览器中的显示效果,如图 12-4 所示。

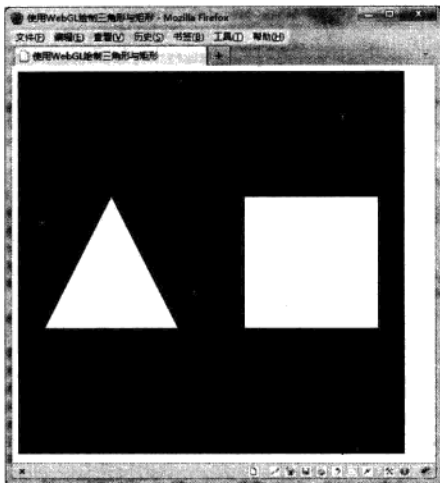


图 12-4 利用 WebGL 绘制一个三角形与一个矩形

### 12.2.3 代码剖析

#### 1. HTML 页面代码

首先介绍示例页面的 HTML 代码部分,该页面中只有一个简单的 canvas 元素,但是在页面打开时调用了 `webGLStart()` 函数。该页面的 HTML 代码如代码清单 12-1 所示。

代码清单 12-1 示例页面的 HTML 代码

```

<html>
<head>
<title> 使用 WebGL 绘制三角形与矩形 </title>
<meta http-equiv="content-type" content="text/html; charset=utf-8">
<script type="text/javascript" src="glMatrix-0.9.5.min.js"></script>
... (JavaScript 脚本代码稍后介绍) ...
</head>
<body onload="webGLStart();" >
<canvas id="canvas1" style="border: none;" width="500"
height="500"></canvas>
</body>
</html>

```

本示例页面中只使用了一个 canvas 元素来显示使用 WebGL 制作出来的图像。在使用 WebGL 制作 3D 图形和动画的时候，都是使用 canvas 元素来进行显示的。

## 2. JavaScript 脚本代码

接下来详细讲解本案例中的 JavaScript 脚本代码部分。

在页面打开时调用了一个 webGLStart 函数，该函数中的代码如下所示。

```

// 绘制 3D 图形
function webGLStart()
{
    var canvas = document.getElementById("canvas1");// 获取 canvas 元素
    initGL(canvas);// 初始化 WebGL
    initShaders();// 初始化渲染器
    initBuffers();// 初始化缓冲区
    // 每次清除 canvas 元素中的内容时均将其填充为黑色
    gl.clearColor(0.0, 0.0, 0.0, 1.0);
    gl.enable(gl.DEPTH_TEST);// 开启深度测试

    drawScene();// 进行绘制
}

```

该函数首先获取页面上的 canvas 元素，然后调用 initGL 函数并将页面上的 canvas 元素作为 initGL 函数的参数来初始化 WebGL 上下文对象，调用 initShaders 函数来初始化渲染器，之后调用 initBuffers 方法初始化几个缓冲区。在本示例中，使用缓冲区来存放将要绘制的三角形与矩形的细节信息。接下来，在代码中进行 WebGL 上下文对象的基本设置，设定每次清除 canvas 元素中的内容时都将该 canvas 元素填充为黑色，然后开启深度测试（在三维场景中，绘制在后面的物体由于被绘制在前面的物体遮盖而不可见）。最后调用 drawScene 函数，利用缓冲区中的信息来绘制图形（三角形与矩形）。

接下来看一下 initGL 函数中的内容。该函数使用页面上的 canvas 元素作为参数，函数功能为初始化 WebGL 上下文对象，代码如下所示。

```

var gl;//WebGL 上下文对象
// 初始化 WebGL 上下文对象
function initGL(canvas)
{
    try
    {
        // 获取 canvas 元素的 WebGL 上下文对象
        gl = canvas.getContext("experimental-webgl");
        // 将 3D 视图的宽度设置为 canvas 元素的宽度
        gl.viewportWidth = canvas.width;
        // 将 3D 视图的高度设置为 canvas 元素的高度
        gl.viewportHeight = canvas.height;
    }
    catch (e)
    {
    }
    if (!gl) // 初始化失败
        alert("对不起, 您不能初始化 WebGL。");
}

```

在函数的开头, 获取了 canvas 元素的 WebGL 上下文对象。在使用 canvas 元素进行 2D 图形图像绘制的时候, 使用 canvas.getContext('2d') 方法获取 canvas 元素的图形上下文对象, 但是在使用 canvas 元素进行 3D 图形、图像和动画绘制的时候, 使用 canvas.getContext("experimental-webgl") 方法获取 canvas 元素的 WebGL 上下文对象, 其中“experimental-webgl”只用于目前 WebGL 尚未完全成熟的阶段, 将来 WebGL 正式版发布后该参数可能会改为“webgl”或者其他名称。在获取 WebGL 上下文对象之后, 就像使用 canvas 元素的图形上下文对象一样, 可以使用 WebGL 上下文对象的各种属性与方法来进行各种 3D 图形、图像的绘制。在这段示例代码中, 首先将 3D 图形、图像的视图尺寸设置为与 canvas 元素的尺寸完全相等。最后, 在代码中实现如果 WebGL 上下文对象初始化失败, 则弹出一个提示信息, 告诉用户不能初始化 WebGL 上下文对象。

接下来, 在查看 initShaders (初始化渲染器) 函数代码之前, 为了更好地理解这个函数, 先来看一下 initBuffers (初始化缓冲区) 函数与 drawScene (绘制图形) 函数中的代码。initBuffers 函数中的代码如下所示。

```

var triangleVertexPositionBuffer;// 三角形顶点位置缓冲区
var squareVertexPositionBuffer;// 矩形顶点位置缓冲区
// 初始化缓冲区
function initBuffers()
{
    // 创建三角形顶点位置缓冲区
    triangleVertexPositionBuffer = gl.createBuffer();
    // 将 triangleVertexPositionBuffer 设定为接下来的操作所使用的缓冲区
    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexPositionBuffer);
    // 使用 JavaScript 列表定义一个等腰三角形的一组顶点信息
    var vertices =
    [

```

```

    0.0, 1.0, 0.0,
    -1.0, -1.0, 0.0,
    1.0, -1.0, 0.0
];
// 使用顶点列表创建 Float32Array 对象并将其填充到缓存区中
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices),
gl.STATIC_DRAW);
// 在缓冲区中存放三个顶点信息，每个顶点由三个数据（三维）构成
triangleVertexPositionBuffer.itemSize = 3;
triangleVertexPositionBuffer.numItems = 3;

// 创建矩形顶点位置缓冲区
squareVertexPositionBuffer = gl.createBuffer();
// 将 squareVertexPositionBuffer 设定为接下来的操作所使用的缓冲区
gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexPositionBuffer);
// 使用 JavaScript 列表定义一个矩形的一组顶点信息
vertices =
[
    1.0, 1.0, 0.0,
    -1.0, 1.0, 0.0,
    1.0, -1.0, 0.0,
    -1.0, -1.0, 0.0
];
// 使用顶点列表创建 Float32Array 对象并将其填充到缓存区中
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices),
gl.STATIC_DRAW);
// 在缓冲区中存放四个顶点信息，每个顶点由三个数据（三维）构成
squareVertexPositionBuffer.itemSize = 3;
squareVertexPositionBuffer.numItems = 4;
}

```

这段代码的开始处定义了两个全局变量，以保存一个存放三角形顶点位置信息的缓冲区与一个存放矩形顶点位置信息的缓冲区，代码如下所示。

```

var triangleVertexPositionBuffer; // 存放三角形顶点数组的缓冲区
var squareVertexPositionBuffer; // 存放矩形数组的缓冲区

```

接下来，在函数中创建一个用来存放三角形顶点位置信息的缓冲区，代码如下所示。

```

// 初始化缓冲区
function initBuffers()
{
    triangleVertexPositionBuffer = gl.createBuffer();
}

```

创建好缓冲区之后，一个缓冲区实际上代表了显卡中的一小块内存。将图形各顶点位置信息先存放在内存中，之后使用 `drawScene` 函数进行绘制的时候，只需通知 WebGL 上下文对象按照缓冲区中存放的各顶点位置信息进行绘制，这样 WebGL 上下文对象就可以自行从缓冲区中（即从内存中）取出预先存放好的各顶点位置信息进行绘制。这种做法可以使代码的运行变得更加高效，尤其是在制作动画（例如让某个图形每秒钟重绘十次以使其产生动画

效果)的时候。当然,在本示例中,只有有限的几个顶点信息,所以在绘制时所花费的成本也相当少,但是在处理比较庞大的图形(比如有1万个顶点)的时候,使用这种方法将真正体现其性能优势。

接下来的一行代码如下所示。

```
gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexPositionBuffer);
```

这行代码的意思是将刚刚创建的 triangleVertexPositionBuffer 缓冲区绑定到一个 buffer 对象上,并且把这个 buffer 对象赋值给 WebGL 上下文对象的 ARRAY\_BUFFER 属性。经过这个绑定与赋值操作之后,接下来绘图时所使用的缓冲区就是 triangleVertexPositionBuffer 缓冲区。

接下来的代码如下所示。

```
var vertices =  
[  
    0.0,  1.0,  0.0,  
    -1.0, -1.0,  0.0,  
    1.0, -1.0,  0.0  
];
```

在此处代码中,利用 JavaScript 的列表创建一个等腰三角形的一组三维顶点的位置信息,它的中心点在(0,0,0)上,然后将这个列表赋值给变量 vertices。

接下来的代码如下所示。

```
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices),  
gl.STATIC_DRAW);
```

此处代码的意思是以 vertices 变量作为参数,创建一个 Float32Array 对象,然后将它填充到目前所使用的 triangleVertexPositionBuffer 缓冲区(被保存在 WebGL 上下文对象的 ARRAY\_BUFFER 属性中)。在后面小节中将对 Float32Array 对象进行介绍,现在只要知道可以使用 Float32Array 对象把 JavaScript 列表(被保存在变量 vertices 中)存放到缓冲区中就可以了。第三个参数用来指定将顶点位置信息存放到缓冲区中后如何进行读取与写入,gl.STATIC\_DRAW 表示将顶点位置信息用于渲染,只需要一次指定缓冲区对象中的顶点位置信息,指定后可以重复使用缓冲区中的各顶点位置信息来进行多次渲染。

接下来的两行代码如下所示。

```
triangleVertexPositionBuffer.itemSize = 3;  
triangleVertexPositionBuffer.numItems = 3;
```

这两行代码的意思是为缓冲区添加两个属性,这两个属性不是 WebGL API 中的内置属性,但会在实际绘图时起到作用。JavaScript 中一个比较有用的特性是可以自行为某个对象添加属性。这两行代码为 triangleVertexPositionBuffer 对象添加了一个 itemSize 属性与一个 numItems 属性,这两个属性的作用是声明 triangleVertexPositionBuffer 对象中有三个顶点

(numItems) 信息, 每个顶点由三个数据 (三维) 构成。

在剩余几行代码中, 使用同样方法创建一个用来存放矩形顶点位置信息的缓冲区, 并且创建一个 JavaScript 列表, 其中存放了一个矩形的一组顶点位置信息, 并将这个 JavaScript 列表填充到存放矩形顶点位置信息的缓冲区中。代码如下所示:

```
squareVertexPositionBuffer = gl.createBuffer(); // 创建矩形顶点位置缓冲区
// 将 squareVertexPositionBuffer 设定为接下来的操作所使用的缓冲区
gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexPositionBuffer);
// 使用 JavaScript 列表定义一个矩形的一组顶点信息
vertices =
[
    1.0, 1.0, 0.0,
    -1.0, 1.0, 0.0,
    1.0, -1.0, 0.0,
    -1.0, -1.0, 0.0
];
// 使用顶点列表创建 Float32Array 对象并填充到缓存区中
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices),
gl.STATIC_DRAW);
// 缓存区中存放四个顶点信息, 每个顶点由三个数据 (三维) 构成
squareVertexPositionBuffer.itemSize = 3;
squareVertexPositionBuffer.numItems = 4;
```

接下来要做的工作就是利用这两个图形的缓冲区, 把图形绘制在屏幕上。drawScene 函数中的代码如下所示。

```
// 绘制图形
function drawScene() {
    // 设置 3D 图像的视图大小
    gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
    // 擦除 canvas 元素中的内容
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
    // 设置对视图的观察视角
    mat4.perspective(45, gl.viewportWidth / gl.viewportHeight, 0.1, 100.0,
    pMatrix);

    // 使用恒等矩阵进行初始化, 将绘制位置设置在视图中央
    mat4.identity(mvMatrix);
    // 将绘制位置左移 1.5 个单位, 后移 7 个单位 (三角形第一个顶点位置处)
    mat4.translate(mvMatrix, [-1.5, 0.0, -7.0]);
    // 将 triangleVertexPositionBuffer 设定为接下来的操作所使用的缓冲区
    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexPositionBuffer);
    /* 通知 WebGL 上下文对象缓冲区中存放的顶点位置信息将作为三角形各顶点位置信息来使用, 每个三角形使用三个数据 (三维信息) */
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
    triangleVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);
    // 通知 WebGL 上下文对象使用当前的模型 - 视图矩阵来绘制图像
    setMatrixUniforms();
    // 绘制三角形
```

```

gl.drawArrays(gl.TRIANGLES, 0, triangleVertexPositionBuffer.numItems);

// 将绘制位置右移 3 个单位
mat4.translate(mvMatrix, [3.0, 0.0, 0.0]);
// 将 squareVertexPositionBuffer 设定为接下来操作所使用的缓冲区
gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexPositionBuffer);
/* 通知 WebGL 上下文对象缓冲区中存放的顶点位置信息将作为矩形各顶点位置信息来使用，
每个矩形使用三个数据（三维信息）*/
gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
squareVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);
// 通知 WebGL 上下文对象使用当前的模型 - 视图矩阵与投影矩阵来绘制图像
setMatrixUniforms();
// 绘制矩形
gl.drawArrays(gl.TRIANGLE_STRIP, 0,
squareVertexPositionBuffer.numItems);}

```

这段代码的第一行如下所示。

```
gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
```

这行代码的意思是将用来显示 3D 图像的视图大小设置为 canvas 元素的大小，每次绘制图形前都必须使用 WebGL 上下文对象来设置显示 3D 图像的视图大小。

接下来的一行代码如下所示。

```
gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
```

这行代码的意思是将 canvas 元素中的内容擦除。

接下来的一行代码如下所示。

```
mat4.perspective(45, gl.viewportWidth / gl.viewportHeight, 0.1, 100.0,
pMatrix);
```

这行代码的意思是设置视图的投影角度。在默认情况下，在使用 WebGL 上下文对象绘制的 3D 图像中，无论是近距离的物体还是远距离的物体，其大小都是一样的（称为垂直投影）。为了使远距离的物体能够看起来小一些，需要设置一个视图的投影角度，在本例中，将投影角度设置为 45 度，同时，将页面中 canvas 元素的宽度与高度的比例通知 WebGL 上下文对象，能够将在该视图中看见的图形的最近距离必须大于 0.1 个单位（默认为像素），最远距离必须在 100 个单位以内的要求通知 WebGL 上下文对象。

在设置视图时，使用一个名为 mat4 的模块的 perspective 方法，同时使用了一个变量 pMatrix，稍后将对 mat4 模块与变量 pMatrix 进行说明，这里只要知道设置视图时需要使用 mat4 模块的方法来进行设置，同时需要使用变量 pMatrix 就可以了。

接下来的一行代码如下所示。

```
mat4.identity(mvMatrix);
```

绘制图形的第一步是将绘制位置移动到 3D 视图的中央。在 OpenGL 中，在进行图形绘

制的时候，需要通知 OpenGL 当前的绘制位置与旋转角度。例如，声明“向前移动 20 个单位，旋转 32 度，然后绘制一个机器人”。这种绘制方式是比较有用的，因为可以把绘制机器人的代码封装在一个函数里，只要在绘制前改变当前位置和当前旋转角度，然后直接调用绘制机器人的函数，就可以在任何位置上使用任何角度来绘制机器人了。

当前位置与当前旋转角度都被保存在一个矩阵中。以前学过矩阵可以用来实现移动、旋转与其他几何变形。可以使用一个  $4 \times 4$  的矩阵来实现任何 3D 空间上的变形操作。在一开始，使用一个恒等矩阵（identity matrix）进行初始化。使用恒等矩阵的含义是不执行任何变形，先将它乘以第一个用来变形的矩阵，再乘以第二个用来变形的矩阵，如果还要执行其他变形，再继续乘以其他用来变形的矩阵，最终得到的矩阵中包含了所有的变形操作，这个矩阵被称为模型－视图矩阵。在本示例中，变量 `mvMatrix` 代表了模型－视图矩阵，而 `mat4.identity` 方法就是使用恒等矩阵来将模型－视图矩阵初始化（绘制位置在视图中央），接下来就可以使用这个矩阵来进行移动与旋转操作了，换句话说，该矩阵将设置绘制图形时的绘制位置。

细心的读者可能已经注意到，在讨论矩阵的时候说的是“在 OpenGL”中，而不是“在 WebGL”中，这是因为 WebGL API 中没有这个功能，本示例使用了一个第三方的矩阵库——`glMatrix`，然后加上了一些 WebGL API 中的功能，才得到本示例中的效果。

接下来的一行代码如下所示。

```
mat4.translate(mvMatrix, [-1.5, 0.0, -7.0]);
```

使用恒等矩阵将变形矩阵初始化，将绘制位置设置在 3D 视图中央后，将绘制位置左移 1.5 个单位（在 X 轴方向上使用负数来设置左移），向后移动 7 个单位（在 Z 轴方向上使用负数来设置向后移动）。

接下来的两行代码如下所示。

```
gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexPositionBuffer);
gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
    triangleVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);
```

前面已经介绍过，为了要使用一个缓冲区，需要调用 WebGL 上下文对象的 `bindBuffer` 方法来指定当前所使用的缓冲区，然后使用该缓冲区来进行有关操作。这里首先指定存放了三角形顶点数组的 `triangleVertexPositionBuffer` 缓冲区作为当前缓冲区。

接下来解释一下 WebGL 上下文对象的 `vertexAttribPointer` 方法及其使用参数的具体含义。

在 WebGL API 中，使用 `vertexAttribPointer` 方法来设置 WebGL 上下文对象在绘图时所使用的每个顶点的顶点信息。该方法有六个参数，第一个参数用来指定一个顶点属性数组，该数组中的每一项用来指定绘制图像时每一个顶点的属性，同时将缓冲区中的每个顶点的顶点位置信息设置为该属性值。第二个参数表示缓冲区中每个顶点位置信息所使用数据的个数（绘制三维图像时每个顶点使用三维信息，所以该值为 3）。第三个参数表示缓冲区中每个



顶点位置信息中所使用数据的数据类型（即三维数据中每个数据的数据类型）。第四个参数表示缓冲区中每个顶点所使用的数据（即三维数据中的每个数据）是否为经过转换处理后的标准的  $[0,1]$  范围内的数据（针对无符号数据类型来说）或标准的  $[-1,1]$  范围内的数据（针对有符号数据类型来说），当参数值为 `false` 时表示使用的数据不使用标准范围内的数据。在 `vertexAttribPointer` 方法中，第四个参数值应该被设定为 `false`。第五个参数表示缓冲区中每个顶点所使用的数据（即三维数据中的每个数据）在内存中是否是连续保存的，通常将该值设定为 0 即可（表示连续保存）。第六个参数表示缓冲区的位置，为 0 时表示使用当前缓冲区来绘制图像（当前缓冲区已经赋值给了 WebGL 上下文对象的 `ARRAY_BUFFER` 属性）。

在本示例中，使用当前缓冲区中所保存的每一个顶点的位置信息来设置顶点属性数组中每一个顶点的属性值（即位置信息），使用 `shaderProgram.vertexPositionAttribute` 来设置绘图时所使用的顶点属性数组，每一个顶点使用三个数据（即 X、Y、Z 轴上的三维信息）。在后面的内容中对 `initShaders`（初始化渲染器）函数进行介绍时将具体介绍本示例中 `shaderProgram.vertexPositionAttribute` 参数值的含义。

接下来的一行代码如下所示。

```
setMatrixUniforms();
```

这行代码的意思是通知 WebGL 上下文对象使用当前的模型 - 视图矩阵（以及投影矩阵，稍后介绍）来进行绘图。这行代码是必需的，因为这个矩阵不是 WebGL API 中内建的矩阵。如果没有该行代码，则所有矩阵变形都会在 JavaScript 脚本内部实现，但不会在视图中起到任何作用，只有通过 `setMatrixUniforms` 函数将变形操作实现到视图中。稍后将对 `setMatrixUniforms` 函数内部代码进行详细讲解。

现在，WebGL 上下文对象已经拥有了一个存放三角形各顶点位置信息的数组，拥有了一个变形矩阵。接下来的一行代码如下所示。

```
gl.drawArrays(gl.TRIANGLES, 0, triangleVertexPositionBuffer.numItems);
```

这行代码的意思是，使用 WebGL 上下文对象所拥有的一个三角形顶点数组来绘制该三角形，从三角形顶点数组的第一个顶点开始绘制，一直到第 `numItems` 个顶点。

在这行代码被执行完毕后，视图中就会显示一个已经绘制好的三角形。

接下来开始绘制矩形。绘制矩形的第一行代码如下所示。

```
mat4.translate(mvMatrix, [3.0, 0.0, 0.0]);
```

首先将当前的绘制位置右移 3 个单位。记住，目前的绘制位置在视图中央向左移 1.5 个单位，内移 7 个单位处，执行了该行代码后，绘制位置在视图中央向右移 1.5 个单位，内移 7 个单位处。

接下来的代码如下所示。

```
gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexPositionBuffer);
gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
```

```
squareVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);
```

这是通知 WebGL 上下文对象使用存放矩形顶点数组的缓冲区作为当前使用的缓冲区，使用缓冲区中存放的矩形顶点数组的每一个顶点的位置信息来设置绘制矩形时所使用的每一个顶点的位置信息，使用该缓冲区的 itemSize 属性来通知 WebGL 上下文对象顶点数组的每一项都使用三个数据（三维信息）。

接下来的一行代码如下所示。

```
setMatrixUniforms();
```

如上所述，该行代码的作用为通知 WebGL 上下文对象使用当前的模型 - 视图矩阵来进行变形操作。

最后，使用如下所示的代码在视图中绘制矩形。

```
gl.drawArrays(gl.TRIANGLE_STRIP, 0, squareVertexPositionBuffer.numItems);
```

读者也许会问：TRIANGLE\_STRIP 是什么？回答是：一系列三角形。在该三角形系列中，使用开头三个顶点定义第一个三角形，再以后两个顶点与一个新顶点定义第二个三角形，接着以后两个顶点与一个新顶点定义第三个三角形，如果还存在新的顶点，则继续定义其他的三角形。在本示例中，这是一种定义矩形的非常快速的方法。在更复杂的情况下，TRIANGLE\_STRIP 可以使用大量三角形来近似定义一个物体的表面。

现在可以将示例程序运行在支持 WebGL API 的浏览器中来查看运行结果，然后尝试修改顶点列表中的顶点信息，尤其是 Z 轴上的坐标点信息，以查看修改之后的结果在浏览器中的运行结果。将矩形在 Z 轴上的坐标点全部改为 2 或 -3，查看当顶点向前移动或向后移动时，是否看起来变得更大（向前移动）或更小（向后移动）了。或者只修改一两个 Z 轴上的坐标点信息，然后观察浏览器中的矩形是否发生了扭曲现象。

接下来回过头来看一下 initShaders（初始化渲染器）函数中的代码。首先来看一下之前提到的模型 - 视图矩阵与投影矩阵的创建方法，代码如下所示。

```
var mvMatrix = mat4.create();
var pMatrix = mat4.create();
```

先定义一个变量 mvMatrix 来代表模型 - 视图矩阵，定义一个变量 pMatrix 来代表投影矩阵。然后利用 mat4.create 方法将这两个矩阵创建出来。创建出来的这两个矩阵的初始状态为空矩阵（内部数据全为 0）。这里介绍一下投影矩阵。在介绍 drawScene 函数的一开始就讲解过，使用 glMatrix 矩阵库中的 mat4.perspective 方法来设置对视图的投影角度，在该方法的参数中使用了变量 pMatrix。使用 glMatrix 矩阵库的原因是 WebGL 并不直接支持投影（就像它并不直接支持模型 - 视图矩阵一样）。就像对图形的移动操作与变形操作是被封装在模型 - 视图矩阵中一样，让远距离的物体看起来比近距离的物体要小得多的处理是被封装在投影矩阵内的。mat4.perspective 方法将投影角度（本示例中为 45 度）、视窗的宽度与高度的比例、最近能看见物体的距离（本示例中为 0.1 个单位）及最远能看见物体的距离（本示例中

为 100 个单位) 这几个参数都封装在了变量 pMatrix 所代表的投影矩阵中。

接下来解释一下什么是渲染器。

以前渲染器被解释成通知系统如何在屏幕上对某个局部进行着色的一段代码。但是, 随着 3D 图像绘制技术的发展, 现在, 渲染器被解释成在图形被绘制前预先能够对所绘制的图形的任何部分进行任何处理的一段程序。渲染器是非常有用的, 因为运行在显卡中, 所以其代码运行起来非常高速, 而且在渲染器中定义图形的变形也是非常方便的。

在这个非常简单的 WebGL 示例程序中使用渲染器的目的是让 WebGL 能够使用渲染器, 并且将渲染器运行在显卡中, 以实现使用视图 - 模型矩阵与投影矩阵在屏幕上绘制图形, 而不需要在 JavaScript 代码中手工绘制每一个图形的顶点与顶点之间的连线。

initShaders (初始化渲染器) 函数中的代码如下所示。

```
var shaderProgram; // 程序对象
// 初始化渲染器
function initShaders()
{
    // 获取片元渲染器
    var fragmentShader = getShader(gl, "shader-fs");
    // 获取顶点渲染器
    var vertexShader = getShader(gl, "shader-vs");
    // 创建程序对象
    shaderProgram = gl.createProgram();
    // 将渲染器绑定到程序对象上
    gl.attachShader(shaderProgram, vertexShader);
    gl.attachShader(shaderProgram, fragmentShader);
    gl.linkProgram(shaderProgram);

    // 如果不能初始化渲染器
    if (!gl.getProgramParameter(shaderProgram, gl.LINK_STATUS)) {
        alert(" 不能初始化渲染器 ");
    }

    gl.useProgram(shaderProgram);

    /* 获取程序对象的 avertexPosition 属性的引用并将其保存在程序对象的
    vertexPosition 属性中 */
    shaderProgram.vertexPositionAttribute =
    gl.getAttribLocation(shaderProgram, "aVertexPosition");
    // 使用程序对象的 vertexPositionAttribute 属性值来绘制顶点位置
    gl.enableVertexAttribArray(shaderProgram.vertexPositionAttribute);
    // 从程序对象中获取两个一致 (uniform) 变量 uPMatrix 与 uVMMatrix 的信息
    shaderProgram.pMatrixUniform = gl.getUniformLocation(shaderProgram,
    "uPMatrix");
    shaderProgram.mvMatrixUniform = gl.getUniformLocation(shaderProgram,
    "uVMMatrix");
}
```

这段代码的开头先通过一个 getShader 函数来获取两个对象: 一个片元渲染器对象与一

个顶点渲染器对象。之后将这两个对象绑定到一个被称为程序（program）的对象上。一个程序对象是一段运行在系统中的，WebGL 上下文对象在绘制图形时所使用的一段代码。这段代码将直接运行在显卡中。通过在程序对象中绑定渲染器对象的方法，可以指定几个渲染器，每个渲染器都将作为程序对象中的一小段代码运行在显卡中。在本示例中，程序对象中绑定了一个片元渲染器与一个顶点渲染器。

创建好程序对象并在其中绑定了片元渲染器与顶点渲染器之后，WegGL 上下文对象将获取对程序对象的一个名为“avertexPosition”的属性的引用，并且将该属性的引用保存在程序对象的 vertexPositionAttribute 属性中。前面介绍过，在 JavaScript 中，可以向对象添加任何属性，在本示例中为程序对象添加了 vertexPosition 属性。获取 avertexPosition 属性的引用并将其保存在程序对象的 vertexPosition 属性中的代码如下所示。

```
shaderProgram.vertexPositionAttribute =
gl.getAttribLocation(shaderProgram, "aVertexPosition");
```

那么，为什么要将 avertexPosition 属性的引用保存在 vertexPositionAttribute 属性中呢？先来看一下 vertexPositionAttribute 属性的作用。

我们曾在 drawScene 函数中使用过这个 vertexPositionAttribute 属性。使用这个属性的代码如下所示。

```
/* 通知 WebGL 上下文对象缓冲区中存放的顶点位置信息将作为三角形各顶点位置信息来使用，
每个三角形使用三个数据（三维信息）*/
gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
triangleVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);
/* 通知 WebGL 上下文对象缓冲区中存放的顶点位置信息将作为矩形各顶点位置信息来使用，
每个三角形使用三个数据（三维信息）*/
gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
squareVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);
```

在上面的代码中，因为程序对象的 vertexPositionAttribute 属性中保存了对 avertexPosition 属性的引用，所以 vertexPositionAttribute 属性实际上起到了一个桥梁作用，将顶点属性数组中已经设置好的各顶点位置信息传递给 avertexPosition 属性，稍后将介绍在渲染器中如何使用 avertexPosition 属性中存放的顶点位置信息来绘制顶点。

接下来，在 initShaders（初始化渲染器）函数中，使用 gl.enableVertexAttribArray 方法通知 WebGL 上下文对象使用程序对象的 vertexPositionAttribute 属性值来绘制顶点位置，代码如下所示。

```
gl.enableVertexAttribArray(shaderProgram.vertexPositionAttribute);
```

initShaders（初始化渲染器）函数中的最后一段代码如下所示。

```
shaderProgram.pMatrixUniform = gl.getUniformLocation(shaderProgram,
"uPMatrix");
shaderProgram.mvMatrixUniform =
gl.getUniformLocation(shaderProgram, "uMVMMatrix");
```

这段代码的含义为：从程序对象中获取两个一致（uniform）变量 uPMatrix 与 uMVMatrix 的定位信息，并将其保存在程序对象的 pMatrixUniform 属性与 mvMatrixUniform 属性中。稍后将对这两个一致变量进行讲解，现在只要知道这两个变量的定位信息被保存在程序对象中就可以了。

接下来看一下 getShader 函数中的代码，如下所示。

```
// 创建渲染器
function getShader(gl, id)
{
    // 根据 id 查找元素
    var shaderScript = document.getElementById(id);
    // 如果找不到元素则返回
    if (!shaderScript)
        return null;
    // 取出元素中所有内容
    var str = "";
    var k = shaderScript.firstChild;
    while (k)
    {
        if (k.nodeType == 3)
            str += k.textContent;
        k = k.nextSibling;
    }

    var shader;
    // 如果元素类型为 "x-shader/x-fragment"
    if (shaderScript.type == "x-shader/x-fragment")
        // 创建片元渲染器
        shader = gl.createShader(gl.FRAGMENT_SHADER);
    // 如果元素类型为 "x-shader/x-vertex"
    else if (shaderScript.type == "x-shader/x-vertex")
        // 创建顶点渲染器
        shader = gl.createShader(gl.VERTEX_SHADER);
    // 如果元素类型为其他
    else
        // 返回 null
        return null;
    // 使用 WebGL 上下文对象编译元素中内容
    gl.shaderSource(shader, str);
    gl.compileShader(shader);
    // 执行错误处理
    if (!gl.getShaderParameter(shader, gl.COMPILE_STATUS))
    {
        alert(gl.getShaderInfoLog(shader));
        return null;
    }
    // 返回渲染器
    return shader;
}
```

这个函数中的代码其实也很简单，其功能主要是使用传入的 id 来寻找元素（id 值为脚本中定义的 shader-fs 渲染器与 shader-vs 渲染器），元素的 id 必须与这个传入的 id 相等，然后取出元素中的内容，根据元素的类型创建片元渲染器或顶点渲染器，并且将其传入显卡中，编译成可以在显卡中运行的形式。接下来，在代码中执行对所有错误的错误处理，然后将渲染器返回。当然，只能在 JavaScript 代码中定义渲染器，而不能将渲染器从 HTML 代码中提取出来，通过这种做法，可以使渲染器更容易读懂，因为渲染器是在页面中使用脚本的形式定义的，就好像它们本身就是一段 JavaScript 脚本一样。

定义这些渲染器的脚本代码如下所示。

```
<script id="shader-fs" type="x-shader/x-fragment">
#ifdef GL_ES
precision highp float;
#endif
void main(void) {
    gl_FragColor = vec4(1.0, 1.0, 1.0, 1.0);
}
</script>
<script id="shader-vs" type="x-shader/x-vertex">
attribute vec3 aVertexPosition;

uniform mat4 uMVMatrix;
uniform mat4 uPMatrix;

void main(void) {
    gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
}
</script>
```

首先需要记住的是，这段代码并不是 JavaScript 脚本代码，尽管这两种脚本语言的祖先是相非常相似的。事实上，以上这段代码是使用一种特殊的称为 GL SL 的渲染器脚本语言书写出来的。该脚本语言来源于 C 语言。第一个渲染器——片元渲染器事实上并不做任何事情，它只是一段必不可少的固定代码，作用是告诉 WebGL 显卡浮点数的精度，以及使用白色来绘制所有图形。第二个渲染器是一个顶点渲染器，是一段运行在显卡中的代码，作用是使用顶点来绘制任何图形。同时，顶点渲染器拥有两个一致变量，分别为变量 uMVMatrix（代表视图模型矩阵）与变量 uPMatrix（代表投影矩阵）。一致变量是非常有用的，因为能够从渲染器外部（实际上是从绑定它们的程序对象之外）来访问它们。我们曾经介绍过，在 initShaders（初始化渲染器）函数中，从程序对象之外获取了一致变量的信息。可以将渲染器的程序对象理解为一个对象，将一致变量理解为一个属性。

现在，绘制每个顶点时都将调用渲染器，并且在 aVertexPosition 属性值中已经存放该顶点的位置信息，因为在 drawScene 函数中，已经利用程序对象的 vertexPosition 属性将 aVertexPosition 属性值设置为缓冲区中设置的该顶点的位置信息。渲染器中主程序的作用是将 aVertexPosition 属性中的顶点位置乘以模型视图矩阵再乘以投影矩阵，得到在屏幕上进行

绘制时该顶点的真正位置。

因此，当 webGLStart 函数调用 initShaders（初始化渲染器）函数的时候，initShaders 函数调用 getShader 函数根据页面中定义的渲染器脚本来装载片元渲染器与顶点渲染器，同时在渲染器脚本中将这两个渲染器编译并传入显卡中，并且利用这两个渲染器在屏幕上进行图形的绘制。

最后，对前面所提到的 setMatrixUniforms 函数中的代码进行讲解。正确理解以上讲解的知识点后，这个函数代码也就好理解了。函数代码如下所示。

```
function setMatrixUniforms() {
    gl.uniformMatrix4fv(shaderProgram.pMatrixUniform, false, pMatrix);
    gl.uniformMatrix4fv(shaderProgram.mvMatrixUniform, false, mvMatrix);
}
```

在 initShaders 函数中，已经从程序对象中获取了一致变量 uPMatrix 与 uMVMatrix 的定位信息，并将它们保存在程序对象的 mvMatrixUniform 属性中。函数 setMatrixUniforms 中的代码的含义为将 JavaScript 类型的 pMatrix 矩阵与 mvMatrix 矩阵传入程序对象的一致变量 uPMatrix 与 uMVMatrix 中。在顶点渲染器的脚本代码的主程序中，可以看到真正绘制顶点时该顶点的三维坐标是通过缓冲区中的顶点位置信息，以及结合了这两个代表投影矩阵与视图－模型矩阵中的投影信息和图形变形信息最终计算出来的三维坐标信息（在 drawScene 函数中设置投影矩阵与视图－模型矩阵中的信息）。

最后完整地看一下本示例的脚本代码，如代码清单 12-2 所示。

代码清单 12-2 本示例的完整脚本代码

---

```
<script type="text/javascript" src="glMatrix-0.9.5.min.js"></script>
<script id="shader-fs" type="x-shader/x-fragment">
#ifdef GL_ES
precision highp float;
#endif
void main(void) {
    gl_FragColor = vec4(1.0, 1.0, 1.0, 1.0);
}
</script>

<script id="shader-vs" type="x-shader/x-vertex">
attribute vec3 aVertexPosition;
uniform mat4 uMVMatrix;
uniform mat4 uPMatrix;
void main(void) {
    gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
}
</script>
<script type="text/javascript">
var gl;//WebGL 上下文对象
// 初始化 WebGL 上下文对象
```

```

function initGL(canvas)
{
    try
    {
        // 获取 canvas 元素的 WebGL 上下文对象
        gl = canvas.getContext("experimental-webgl");
        // 将 3D 视图的宽度设置为 canvas 元素的宽度
        gl.viewportWidth = canvas.width;
        // 将 3D 视图的高度设置为 canvas 元素的高度
        gl.viewportHeight = canvas.height;
    }
    catch (e)
    {
    }
    if (!gl) // 初始化失败
        alert("对不起, 您不能初始化 WebGL。");
}

// 创建渲染器
function getShader(gl, id)
{
    // 根据 id 查找页面上的元素
    var shaderScript = document.getElementById(id);
    // 如果找不到元素则返回
    if (!shaderScript)
        return null;
    // 取出元素中所有内容
    var str = "";
    var k = shaderScript.firstChild;
    while (k)
    {
        if (k.nodeType == 3)
            str += k.textContent;
        k = k.nextSibling;
    }

    var shader;
    // 如果元素类型为 "x-shader/x-fragment"
    if (shaderScript.type == "x-shader/x-fragment")
        // 创建片元渲染器
        shader = gl.createShader(gl.FRAGMENT_SHADER);
    // 如果元素类型为 "x-shader/x-vertex"
    else if (shaderScript.type == "x-shader/x-vertex")
        // 创建顶点渲染器
        shader = gl.createShader(gl.VERTEX_SHADER);
    // 如果元素类型为其他
    else
        // 返回 null
        return null;
    // 使用 WebGL 上下文对象编译元素中内容
    gl.shaderSource(shader, str);

```



```

    gl.compileShader(shader);
    // 执行错误处理
    if (!gl.getShaderParameter(shader, gl.COMPILE_STATUS))
    {
        alert(gl.getShaderInfoLog(shader));
        return null;
    }
    // 返回渲染器
    return shader;
}

var shaderProgram;// 程序对象
// 初始化渲染器
function initShaders()
{
    // 获取片元渲染器
    var fragmentShader = getShader(gl, "shader-fs");
    // 获取顶点渲染器
    var vertexShader = getShader(gl, "shader-vs");
    // 创建程序对象
    shaderProgram = gl.createProgram();
    // 将渲染器绑定到程序对象上
    gl.attachShader(shaderProgram, vertexShader);
    gl.attachShader(shaderProgram, fragmentShader);
    gl.linkProgram(shaderProgram);

    // 如果不能初始化渲染器
    if (!gl.getProgramParameter(shaderProgram, gl.LINK_STATUS))
        alert(" 不能初始化渲染器 ");

    gl.useProgram(shaderProgram);
    /* 获取对程序对象的 aVertexPosition 属性的引用并将其保存在程序对象的
    vertexPosition 属性中 */
    shaderProgram.vertexPositionAttribute =
    gl.getAttribLocation(shaderProgram, "aVertexPosition");
    // 使用程序对象的 vertexPositionAttribute 属性值来绘制顶点位置
    gl.enableVertexAttribArray(shaderProgram.vertexPositionAttribute);
    // 从程序对象中获取两个一致 (uniform) 变量 uPMatrix 与 uVMMatrix 的信息
    shaderProgram.pMatrixUniform = gl.getUniformLocation(shaderProgram,
    "uPMatrix");
    shaderProgram.mvMatrixUniform = gl.getUniformLocation(shaderProgram,
    "uVMMatrix");
}

var mvMatrix = mat4.create();
var pMatrix = mat4.create();

function setMatrixUniforms()
{
    gl.uniformMatrix4fv(shaderProgram.pMatrixUniform, false, pMatrix);
}

```

```

gl.uniformMatrix4fv(shaderProgram.mvMatrixUniform, false, mvMatrix);
}

var triangleVertexPositionBuffer; // 三角形顶点位置缓冲区
var squareVertexPositionBuffer; // 矩形顶点位置缓冲区
// 初始化缓冲区
function initBuffers()
{
    // 创建三角形顶点位置缓冲区
    triangleVertexPositionBuffer = gl.createBuffer();
    // 将 triangleVertexPositionBuffer 设定为接下来的操作所使用的缓冲区
    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexPositionBuffer);
    // 使用 JavaScript 列表定义一个等腰三角形的一组顶点信息
    var vertices =
    [
        0.0, 1.0, 0.0,
        -1.0, -1.0, 0.0,
        1.0, -1.0, 0.0
    ];
    // 使用顶点列表创建 Float32Array 对象并填充到缓冲区中
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices),
    gl.STATIC_DRAW);
    // 缓冲区中存放三个顶点信息，每个顶点由三个数据（三维）构成
    triangleVertexPositionBuffer.itemSize = 3;
    triangleVertexPositionBuffer.numItems = 3;

    // 创建矩形顶点位置缓冲区
    squareVertexPositionBuffer = gl.createBuffer();
    // 将 squareVertexPositionBuffer 设定为接下来的操作所使用的缓冲区
    gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexPositionBuffer);
    // 使用 JavaScript 列表定义一个矩形的一组顶点信息
    vertices =
    [
        1.0, 1.0, 0.0,
        -1.0, 1.0, 0.0,
        1.0, -1.0, 0.0,
        -1.0, -1.0, 0.0
    ];
    // 使用顶点列表创建 Float32Array 对象并填充到缓冲区中
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices),
    gl.STATIC_DRAW);
    // 缓冲区中存放四个顶点信息，每个顶点由三个数据（三维）构成
    squareVertexPositionBuffer.itemSize = 3;
    squareVertexPositionBuffer.numItems = 4;
}

// 绘制图形
function drawScene()
{
    // 设置 3D 视图的视图大小
    gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);

```

```

// 擦除 canvas 元素中内容
gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
// 设置对视图的观察视角
mat4.perspective(45, gl.viewportWidth / gl.viewportHeight, 0.1, 100.0,
pMatrix);
// 使用恒等矩阵进行初始化, 将绘制位置设置在视图中央
mat4.identity(mvMatrix);
// 将绘制位置左移 1.5 个单位, 后移 7 个单位 (三角形第一个顶点位置处)
mat4.translate(mvMatrix, [-1.5, 0.0, -7.0]);
// 将 triangleVertexPositionBuffer 设定为接下来的操作所使用的缓冲区
gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexPositionBuffer);
/* 通知 WebGL 上下文对象缓冲区中存放的顶点位置信息将作为三角形各顶点位置信息来使用, 每个三角形使用三个数据 (三维信息) */
gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
triangleVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);
// 通知 WebGL 上下文对象使用当前的模型 - 视图矩阵与投影矩阵
setMatrixUniforms();
// 绘制三角形
gl.drawArrays(gl.TRIANGLES, 0, triangleVertexPositionBuffer.numItems);

// 将绘制位置右移 3 个单位
mat4.translate(mvMatrix, [3.0, 0.0, 0.0]);
// 将 squareVertexPositionBuffer 设定为接下来的操作所使用的缓冲区
gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexPositionBuffer);
/* 通知 WebGL 上下文对象缓冲区中存放的顶点位置信息将作为矩形各顶点位置信息来使用, 每个矩形使用三个数据 (三维信息) */
gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
squareVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);
// 通知 WebGL 上下文对象使用当前的模型 - 视图矩阵与投影矩阵
setMatrixUniforms();
// 绘制矩形
gl.drawArrays(gl.TRIANGLE_STRIP, 0,
squareVertexPositionBuffer.numItems);
}
// 绘制 3D 图形
function WebGLStart()
{
    var canvas = document.getElementById("canvas1");// 获取 canvas 元素
    initGL(canvas);// 初始化 WebGL
    initShaders();// 初始化渲染器
    initBuffers();// 初始化缓冲区
    // 每次清除 canvas 元素中的内容时均将其填充为黑色
    gl.clearColor(0.0, 0.0, 0.0, 1.0);
    gl.enable(gl.DEPTH_TEST);// 使能深度测试

    drawScene();// 进行绘制
}
</script>

```

## 12.3 使用颜色绘制彩色三角形与矩形

本节介绍如何在 3D 视图中使用颜色来绘制彩色图形。

### 12.3.1 画面式样

本节将结合一个绘制彩色图形的示例向读者介绍如何在 3D 视图中使用颜色来绘制图形。本示例程序以 12.2 节的示例程序为基础，在屏幕中绘制一个彩色的三角形与矩形。示例程序在浏览器中的显示效果如图 12-5 所示。

### 12.3.2 代码剖析

本节的示例程序的 HTML 代码部分与 12.2 节中示例程序的 HTML 代码部分完全一致，故不再赘述。接下来主要看一下示例程序中的脚本代码部分。

本示例程序的脚本代码也与代码清单 12-2 中的内容大致相同，包括以下几个内容：

1) 使用 `<script>` 标签，并且将该标签的 `type` 属性值分别设定为“`x-shader/x-vertex`”与“`x-shader/x-fragment`”，然后在标签中分别定义顶点渲染器与片元渲染器。

2) 使用 `initGL` 函数初始化 WebGL 上下文对象。

3) 使用 `initShaders` 函数来初始化渲染器，在该函数中调用 `getShader` 函数，在 `getShader` 函数中创建渲染器。

4) 定义模型—视图矩阵 `mvMatrix` 与投影矩阵 `pMatrix`，使用 `setMatrixUniforms` 函数将 JavaScript 类型的矩阵传入到程序对象的一致变量 `uPMatrix` 与 `uVMMatrix` 中，从而使渲染器可以使用这两个矩阵。

5) 利用 `initBuffers` 函数初始化缓冲区，并且在缓冲区中设置好三角形与矩形的各顶点信息。

6) 使用 `drawScene` 函数来绘制三角形与矩形。

7) 定义 `webGLStart` 函数以实现在屏幕上绘制 3D 图形。

本示例程序做出的少量修改是对渲染器的修改、对 `initBuffers` 函数的修改以及对 `drawScene` 函数的修改。要很好地理解这些修改，需要知道一些 WebGL 的渲染管道的知识。图 12-6 是该渲染管道的一个流程图。

这个流程图以十分简单的形式展示了在 `drawScene` 函数中传入 JavaScript 函数中的数据最终变成屏幕上 `canvas` 元素中显示的像素的一个过程。图 12-6 只展示了要掌握本节内容所需要理解的几个步骤。在后面几节中将深入讨论渲染管道的细节部分。

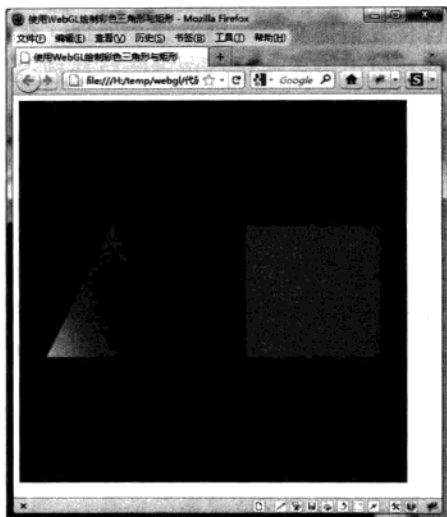


图 12-5 绘制彩色三角形与矩形

在最顶端的处理为：每次调用诸如 `drawArrays`（绘制单个图形所用函数）这样的函数时，WebGL 图形上下文对象都要先处理以属性（例如代码清单 12-2 的程序绘制顶点时所要用到的顶点属性数组中的各顶点属性）和一致变量（例如代表投影矩阵和模型 - 视图矩阵的一致变量）的形式提交给 WebGL 图形上下文对象的数据，并且将这些数据提交给顶点渲染器。

绘制每一个顶点时都会调用顶点渲染器，每次调用时都会先执行将数据提交给渲染器这个处理，在执行这个处理前会先在顶点属性数组中设置好每个顶点的包括顶点位置信息在内的各种属性，同时传入一致变量，然后渲染管道将每个顶点的属性与一致变量提交给顶点渲染器。接下来顶点渲染器将会处理这些数据，会使用到投影矩阵与模型 - 视图矩阵，所以要通过三维投影角度来观察这些顶点，同时渲染器将根据模型 - 视图矩阵中的数据来移动绘图位置和设置旋转角度。接下来，渲染器将这些数据的处理结果保存在易变变量中。渲染器可以输出许多易变变量，其中易变变量 `gl_Position` 是必须输出的，该变量中包含了渲染结束后顶点的最终坐标。

当顶点渲染器中的处理结束后，WebGL 上下文对象根据易变变量中的值把 3D 图像转变成成为 2D 图像，然后为绘制 2D 图像中的每一个像素而调用片元渲染器（因此在有些 3D 图形系统中，片元渲染器也被看成是像素渲染器）。当然，这意味着 WebGL 上下文对象只为不是顶点的像素调用片元渲染器，或者说只为顶点与顶点之间的像素调用片元渲染器。同时，WebGL 上下文对象使用一种叫做线性插值的方法在由图形的顶点与顶点所组成的封闭区域的内部使用像素点对象（在程序语言中被称为 `point` 对象）来进行填充。使用片元渲染器的作用是可以过一个名为 `gl_FragColor` 的易变变量来设置与返回每一个插入的像素点的颜色。

片元渲染器中的处理（在图形内部使用像素点对象进行填充）完成后，通过 WebGL 上下文对象将处理结果进行混合，然后将最终处理结果放入帧缓冲区中，最终将这些帧缓冲区中的内容显示在屏幕上。

现在可以知道，本节一个非常重要的技巧就是如何从 JavaScript 代码中获取图形中各顶点的颜色，并且将其传送到片元渲染器中。

这个技巧的处理方法是：将许多易变变量，不仅仅是顶点的位置信息，从顶点渲染器传送到片元渲染器中。这样可以先将颜色值作为顶点属性数组中各顶点的属性传送到顶点渲染器中，再通过易变变量传送到片元渲染器中。

通过这种处理方法，可以很方便地实现对渐变颜色的处理。因为在片元渲染器中，图形中各顶点之间的像素点的绘制都是通过线性插值的方法来实现的。通过这种插值的方法，可以实现颜色与颜色之间的平滑渐变，就像图 12-5 中三角形内部颜色的平滑渐变那样。

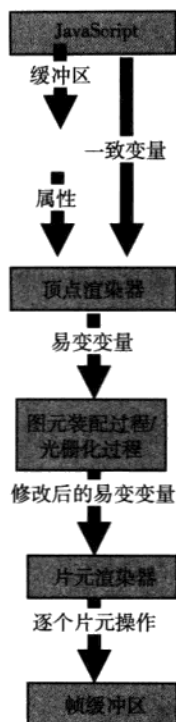


图 12-6 WebGL 渲染管道的流程图

接下来依次看一下本节对代码清单 12-2 中作了哪些改变。首先来看一下顶点渲染器中的代码，该渲染器中代码的改动比较大，如下所示。

```
attribute vec3 aVertexPosition;
attribute vec4 aVertexColor;

uniform mat4 uMVMMatrix;
uniform mat4 uPMatrix;

varying vec4 vColor;

void main(void) {
    gl_Position = uPMatrix * uMVMMatrix * vec4(aVertexPosition, 1.0);
    vColor = aVertexColor;
}
```

现在顶点渲染器中拥有了两个属性——aVertexPosition 属性与 aVertexColor 属性（在每一个顶点中这两个属性的属性值都是不一样的），两个一致变量——uMVMMatrix（代表模型-视图矩阵）与 uPMatrix（代表投影矩阵），以及一个输出值——易变变量 vColor。

在渲染器的主程序中，通过与代码清单 12-2 中相同的方法计算出了变量 gl\_Position（该变量被隐式定义成每个顶点中都有的易变变量）的值。而对于颜色值的处理，将直接从 aVertexColor 属性中读取，并且直接输出到易变变量 vColor 中。

一旦每个顶点都调用渲染器并完成渲染器中的处理后，就会执行片元渲染器中的线性插值处理来生成每个顶点所形成的图形内部的所有像素。

接下来看一下本示例的片元渲染器中的代码，如下所示。

```
#ifdef GL_ES
precision highp float;
#endif

varying vec4 vColor;

void main(void)
{
    gl_FragColor = vColor;
}
```

在这段代码中，使用顶点渲染器中传入的 vColor 这个易变变量，在线性插值处理完毕之后，这个易变变量中的颜色值已经是经过线性插值时所作的颜色渐变处理之后的颜色值了。在片元渲染器的主程序中，将这个经过渐变处理的颜色值作为每一个片元（默认为像素）的颜色值返回。

除了在顶点渲染器与片元渲染器的代码中进行了修改之外，本示例程序还在代码清单 12-2 的其他两个地方作了修改。其中第一个修改比较小，就是在 initShaders 函数中，从原来只获取 aVertexPosition 这一个属性的引用修改为获取两个属性的引用，添加对 aVertexColor

这一属性的引用，并且将其保存在程序对象的 vertexColor 属性中，代码如下所示。

```
shaderProgram.vertexColorAttribute = gl.getAttribLocation(shaderProgram,
    "aVertexColor");
gl.enableVertexAttribArray(shaderProgram.vertexColorAttribute);
```

修改后的 initShaders 函数中的完整代码如下所示。

```
function initShaders()
{
    // 获取片元渲染器
    var fragmentShader = getShader(gl, "shader-fs");
    // 获取顶点渲染器
    var vertexShader = getShader(gl, "shader-vs");
    // 创建程序对象
    shaderProgram = gl.createProgram();
    // 将渲染器绑定到程序对象上
    gl.attachShader(shaderProgram, vertexShader);
    gl.attachShader(shaderProgram, fragmentShader);
    gl.linkProgram(shaderProgram);

    // 如果不能初始化渲染器
    if (!gl.getProgramParameter(shaderProgram, gl.LINK_STATUS))
        alert("不能初始化渲染器");

    gl.useProgram(shaderProgram);
    /* 获取程序对象的 aVertexPosition 属性的引用并将其保存在程序对象的
    vertexPosition 属性中 */
    shaderProgram.vertexPositionAttribute =
    gl.getAttribLocation(shaderProgram, "aVertexPosition");
    // 使用程序对象的 vertexPositionAttribute 属性值来绘制顶点位置
    gl.enableVertexAttribArray(shaderProgram.vertexPositionAttribute);

    /* 获取程序对象的 aVertexColor 属性的引用并将其保存在程序对象的
    vertexColor 属性中 */
    shaderProgram.vertexColorAttribute =
    gl.getAttribLocation(shaderProgram,
    "aVertexColor");
    // 使用程序对象的 vertexColorAttribute 属性值来绘制顶点颜色
    gl.enableVertexAttribArray(shaderProgram.vertexColorAttribute);

    // 从程序对象中获取两个一致 (uniform) 变量 uPMatrix 与 uVMMatrix 的信息
    shaderProgram.pMatrixUniform = gl.getUniformLocation(shaderProgram,
    "uPMatrix");
    shaderProgram.mvMatrixUniform = gl.getUniformLocation(shaderProgram,
    "uVMMatrix");
}
```

本示例程序对代码清单 12-2 的第二个修改是：在 initBuffers（初始化缓冲区）中，不仅为三角形与矩形的各顶点位置信息使用缓冲区，还为这两个图形的各顶点颜色信息使用缓冲区，同时还要修改 drawScene（绘制图形）函数，使 WebGL 上下文对象利用缓冲区中的各项

点颜色信息进行图形的绘制。

首先来看一下 `initBuffers` 函数。在该函数中先定义两个分别代表三角形各顶点颜色信息的缓冲区与代表矩形各顶点颜色信息的缓冲区的全局变量，代码如下所示。

```
var triangleVertexColorBuffer; // 三角形各顶点颜色信息缓冲区
var squareVertexColorBuffer; // 矩形各顶点颜色信息缓冲区
```

接下来要在 `initBuffers` 函数内部添加创建三角形各顶点颜色信息的缓冲区与矩形各顶点颜色信息的缓冲区的代码，同时添加指定三角形各顶点颜色信息与指定矩形各顶点颜色信息的代码。创建三角形各顶点颜色信息的缓冲区并指定三角形各顶点颜色信息的代码如下所示。

```
// 创建三角形各顶点颜色信息的缓冲区
triangleVertexColorBuffer = gl.createBuffer();
// 将该缓冲区指定为当前操作所用缓冲区
gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexColorBuffer);
// 使用 JavaScript 列表定义三角形各顶点所用颜色
var colors =
[
    1.0, 0.0, 0.0, 1.0,
    0.0, 1.0, 0.0, 1.0,
    0.0, 0.0, 1.0, 1.0
];
// 使用顶点颜色列表创建 Float32Array 对象填充缓存
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(colors), gl.STATIC_DRAW);
// 缓冲区中存放三个顶点信息，每个顶点由四个数据构成
triangleVertexColorBuffer.itemSize = 4;
triangleVertexColorBuffer.numItems = 3;
```

由代码注释可知，利用 JavaScript 列表来定义三角形各顶点所用颜色信息，每一个顶点都使用列表中的一行（四个）数据，就像在使用 JavaScript 列表来定义三角形各顶点位置信息时每一个顶点使用列表中的三个数据那样。区别在于：定义顶点位置信息时每个顶点使用列表中的三个数据，这三个数据分别为 X 坐标值、Y 坐标值与 Z 坐标值；定义顶点颜色信息时每个顶点使用列表中的四个数据，分别为 red（红色）值、green（绿色）值、blue（蓝色）值与 alpha（透明度）值。在指定 alpha 值的时候 0 代表完全透明，1 代表完全不透明。同时缓冲区的 `itemSize` 值也从 3 变成了 4。

接下来看一下创建矩形各顶点颜色信息的缓冲区并指定矩形各顶点颜色信息的代码，这一次为每个顶点使用相同的颜色，因此通过循环来指定颜色值，代码如下所示。

```
// 创建矩形各顶点颜色信息的缓冲区
squareVertexColorBuffer = gl.createBuffer();
// 将该缓冲区指定为当前操作所用缓冲区
gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexColorBuffer);
// 使用 JavaScript 列表定义矩形各顶点所用颜色
colors = []
```



```

for (var i=0; i < 4; i++)
{
    colors = colors.concat([0.5, 0.5, 1.0, 1.0]);
}
// 使用顶点颜色列表创建 Float32Array 对象填充缓存
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(colors), gl.STATIC_DRAW);
// 缓冲区中存放四个顶点信息, 每个顶点由四个数据构成
squareVertexColorBuffer.itemSize = 4;
squareVertexColorBuffer.numItems = 4;

```

initBuffers 函数中的完整代码如下所示。

```

var triangleVertexPositionBuffer; // 三角形顶点位置缓冲区
var triangleVertexColorBuffer; // 三角形各顶点颜色信息缓冲区
var squareVertexPositionBuffer; // 矩形顶点位置缓冲区
var squareVertexColorBuffer; // 矩形各顶点颜色信息缓冲区
// 初始化缓冲区
function initBuffers()
{
    // 创建三角形顶点位置缓冲区
    triangleVertexPositionBuffer = gl.createBuffer();
    // 将 triangleVertexPositionBuffer 设定为接下来的操作所使用的缓冲区
    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexPositionBuffer);
    // 利用 JavaScript 列表定义一个等腰三角形的一组顶点信息
    var vertices =
    [
        0.0, 1.0, 0.0,
        -1.0, -1.0, 0.0,
        1.0, -1.0, 0.0
    ];
    // 利用顶点列表创建 Float32Array 对象填充缓存
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices),
    gl.STATIC_DRAW);
    // 缓冲区中存放三个顶点信息, 每个顶点由三个数据 (三维) 构成
    triangleVertexPositionBuffer.itemSize = 3;
    triangleVertexPositionBuffer.numItems = 3;

    // 创建三角形各顶点颜色信息的缓冲区
    triangleVertexColorBuffer = gl.createBuffer();
    // 将该缓冲区指定为当前操作所用缓冲区
    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexColorBuffer);
    // 利用 JavaScript 列表定义三角形各顶点所用颜色
    var colors =
    [
        1.0, 0.0, 0.0, 1.0,
        0.0, 1.0, 0.0, 1.0,
        0.0, 0.0, 1.0, 1.0
    ];
    // 利用顶点颜色列表创建 Float32Array 对象填充缓存
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(colors),
    gl.STATIC_DRAW);
}

```

```

// 缓冲区中存放三个顶点信息, 每个顶点由四个数据构成
triangleVertexColorBuffer.itemSize = 4;
triangleVertexColorBuffer.numItems = 3;

// 创建矩形顶点位置缓冲区
squareVertexPositionBuffer = gl.createBuffer();
// 将 squareVertexPositionBuffer 设定为接下来的操作所使用的缓冲区
gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexPositionBuffer);
// 使用 JavaScript 列表定义一个矩形的一组顶点信息
vertices =
[
    1.0,  1.0,  0.0,
    -1.0,  1.0,  0.0,
    1.0, -1.0,  0.0,
    -1.0, -1.0,  0.0
];
// 利用顶点列表创建 Float32Array 对象填充缓存
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices),
gl.STATIC_DRAW);
// 缓冲区中存放四个顶点信息, 每个顶点由三个数据(三维)构成
squareVertexPositionBuffer.itemSize = 3;
squareVertexPositionBuffer.numItems = 4;

// 创建矩形各顶点颜色信息的缓冲区
squareVertexColorBuffer = gl.createBuffer();
// 将该缓冲区指定为当前操作所用缓冲区
gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexColorBuffer);
// 利用 JavaScript 列表定义矩形各顶点所用颜色
colors = []
for (var i=0; i < 4; i++)
{
    colors = colors.concat([0.5, 0.5, 1.0, 1.0]);
}
// 利用顶点颜色列表创建 Float32Array 对象填充缓存
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(colors),
gl.STATIC_DRAW);
// 缓冲区中存放四个顶点信息, 每个顶点由四个数据构成
squareVertexColorBuffer.itemSize = 4;
squareVertexColorBuffer.numItems = 4;
}

```

最后看一下在 drawScene 函数中所做的修改。在该函数中添加代码, 使其能够使用三角形与矩形的各顶点颜色信息缓冲区中的颜色信息进行三角形与矩形的绘制, 添加的代码如下所示。

```

// 将 triangleVertexColorBuffer 设定为接下来的操作所使用的缓冲区
gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexColorBuffer);
/* 通知 WebGL 上下文对象缓冲区中存放的顶点颜色信息将作为三角形各顶点颜色信息来使用,
每个三角形使用四个数据 */
gl.vertexAttribPointer(shaderProgram.vertexColorAttribute,
triangleVertexColorBuffer.itemSize, gl.FLOAT, false, 0, 0);

```

```
// 将 squareVertexColorBuffer 设定为接下来的操作所使用的缓冲区
gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexColorBuffer);
/* 通知 WebGL 上下文对象缓冲区中存放的顶点颜色信息将被作为矩形各顶点颜色信息来使用，
每个矩形使用四个数据 */
gl.vertexAttribPointer(shaderProgram.vertexColorAttribute,
squareVertexColorBuffer.itemSize, gl.FLOAT, false, 0, 0);
```

drawScene 函数中的完整代码如下所示。

```
// 绘制图形
function drawScene()
{
    // 设置 3D 视图的视图大小
    gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
    // 擦除 canvas 元素中内容
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
    // 设置对视图的观察视角
    mat4.perspective(45, gl.viewportWidth / gl.viewportHeight, 0.1, 100.0,
    pMatrix);
    // 使用恒等矩阵进行初始化，将绘制位置设置在视图中央
    mat4.identity(mvMatrix);
    // 将绘制位置左移 1.5 个单位，后移 7 个单位（三角形第一个顶点位置处）
    mat4.translate(mvMatrix, [-1.5, 0.0, -7.0]);
    // 将 triangleVertexPositionBuffer 设定为接下来的操作所使用的缓冲区
    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexPositionBuffer);
    /* 通知 WebGL 上下文对象缓冲区中存放的顶点位置信息将作为三角形各顶点位置信息来使用，
    每个三角形使用三个数据（三维信息）*/
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
    triangleVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);
    // 将 triangleVertexColorBuffer 设定为接下来的操作所使用的缓冲区
    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexColorBuffer);
    /* 通知 WebGL 上下文对象缓冲区中存放的顶点颜色信息将作为三角形各顶点颜色信息来使用，
    每个三角形使用四个数据 */
    gl.vertexAttribPointer(shaderProgram.vertexColorAttribute,
    triangleVertexColorBuffer.itemSize, gl.FLOAT, false, 0, 0);

    // 通知 WebGL 上下文对象使用当前的模型 - 视图矩阵与投影矩阵
    setMatrixUniforms();
    // 绘制三角形
    gl.drawArrays(gl.TRIANGLES, 0, triangleVertexPositionBuffer.numItems);

    // 将绘制位置右移 3 个单位
    mat4.translate(mvMatrix, [3.0, 0.0, 0.0]);
    // 将 squareVertexPositionBuffer 设定为接下来的操作所使用的缓冲区
    gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexPositionBuffer);
    /* 通知 WebGL 上下文对象缓冲区中存放的顶点位置信息将作为矩形各顶点位置信息来使用，
    每个矩形使用三个数据（三维信息）*/
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
    squareVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);
    // 将 squareVertexColorBuffer 设定为接下来的操作所使用的缓冲区
    gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexColorBuffer);
```

```

/* 通知 WebGL 上下文对象缓冲区中存放的顶点颜色信息将作为矩形各顶点颜色信息来使用，
每个矩形使用四个数据 */
gl.vertexAttribPointer(shaderProgram.vertexColorAttribute,
squareVertexColorBuffer.itemSize, gl.FLOAT, false, 0, 0);
// 通知 WebGL 上下文对象使用当前的模型 - 视图矩阵与投影矩阵
setMatrixUniforms();
// 使用 WebGL 上下文对象绘制矩形
gl.drawArrays(gl.TRIANGLE_STRIP, 0,
squareVertexPositionBuffer.numItems);
}

```

## 12.4 制作三维动画

本节介绍如何使用 WebGL 上下文对象来制作三维动画。

### 12.4.1 画面式样

本节将通过一个使用 WebGL 上下文对象绘制动画的案例向读者介绍如何在 3D 视图使用 WebGL 上下文对象来绘制动画。本示例程序以 12.3 节的示例程序为基础，使屏幕中的三角形与矩形具有动画功能。示例程序在浏览器中的显示效果如图 12-7 所示。

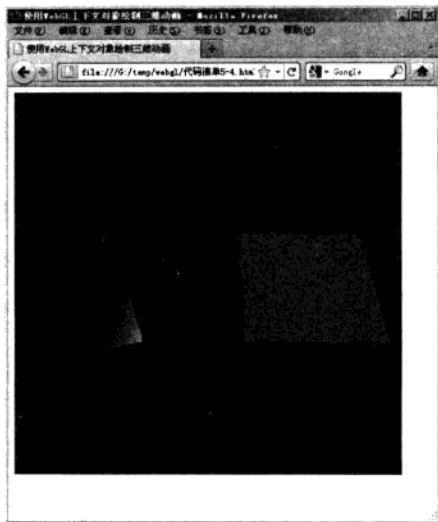


图 12-7 使用 WebGL 上下文对象绘制三维动画

### 12.4.2 代码剖析

首先介绍的是，通过 WebGL 上下文对象的使用，可以很轻松地进行 3D 动画的绘制工作。要不断地使用不同的参数（如绘制位置和图形大小等参数）重绘 3D 图像。对于这个重绘的概念，很多读者可能误认为它的工作原理是“在 X 点处绘制一个图形，下一次绘制动画时将该图形移动到 Y 点，再下一次绘制动画时将该图形移动到 Z 点……”，但是重绘的真正工作原理应该是“在 X 点处绘制一个图形，下一次绘制动画时在 Y 点处重新绘制该图形，再下一次绘制动画时在 Z 点处重新绘制该图形……”。

为什么重绘的工作原理是每一次都要重绘图形呢？这是因为 WebGL 上下文对象使用一个 drawScene 函数来绘制图形。在制作动画时每一次重绘都需要调用该函数，然后使用不同的方法来重绘图形。

接着先看一下本节示例程序中 webGLStart 函数的代码，它在 12.3 节示例程序中 webGLStart 函数的基础上进行了修改。注意，在函数底部并不是直接调用 drawScene 函数来绘制图形，而是调用了 tick 函数，在该函数中调用 drawScene 函数来绘制图形。webGLStart 函数的代码如下所示。

```
// 绘制 3D 图形
function webGLStart()
{
    var canvas = document.getElementById("canvas1");// 获取 canvas 元素
    initGL(canvas);// 初始化 WebGL 上下文对象
    initShaders();// 初始化渲染器
    initBuffers();// 初始化缓冲区
    // 每次清除 canvas 元素中的内容时均将其填充为黑色
    gl.clearColor(0.0, 0.0, 0.0, 1.0);
    gl.enable(gl.DEPTH_TEST);// 开启深度测试
    tick();// 调用 tick 函数
}
```

在 tick 函数中使用 WebGL API 中的动画功能，每一次使用不同的方法来绘制图形（例如将三角形的旋转角度从 81 度变为 82 度），同时要在该函数中指定一个每次调用 drawScene 函数重绘图形的间隔时间。

接下来看一下 tick 函数中的代码，该函数中的代码如下所示。

```
function tick()
{
    requestAnimationFrame(tick);
    drawScene();
    animate();
}
```

第一行代码中的 requestAnimationFrame 函数是本节示例程序中所引用的 Google 所提供的一个脚本文件 webgl-utils.js 中的一个函数，该脚本文件的引用方法如下所示。

```
<script type="text/javascript" src="webgl-utils.js"></script>
```

requestAnimationFrame 函数的作用相当于 JavaScript 中的 setInterval 函数，通知 WebGL 上下文对象每次重绘完图形后都调用 tick 函数再次进行图形的重绘。但是区别在于，对于 setInterval 函数，如果在浏览器的一个标签中打开使用 setInterval 函数的页面，然后在浏览器的另一个标签中打开其他页面，那么即使用 setInterval 函数的页面被隐藏起来，其中的 setInterval 函数仍然处于工作状态，每隔一段时间依然会执行图形的重绘。这显然是一件不好的事情。使用 requestAnimationFrame 函数的好处在于，只有当标签处于显示状态时才会在每隔一段时间后执行图形的重绘工作。

接下来看一下 drawScene 函数中的代码，讲解完该函数之后，再回过头来看一下 tick 函数代码的第三行中的 animate 函数。

在示例程序的脚本文件中，在 drawScene 函数之前先定义了两个全局变量，如下所示。

```
var rTri = 0;// 三角形的旋转角度
var rSquare = 0; // 矩形的旋转角度
```

这两个变量分别代表三角形的旋转角度与矩形的旋转角度，初始值均为 0，在每次进行图形重绘时旋转角度均会增加 1 度，从而使图形旋转起来。

接下来看一下示例程序中 drawScene 函数的完整代码，如下所示。

```
var rTri = 0; // 三角形的旋转角度
var rSquare = 0; // 矩形的旋转角度
// 绘制图形
function drawScene()
{
    // 设置 3D 视图的视图大小
    gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
    // 擦除 canvas 元素中内容
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
    // 设置对视图的观察视角
    mat4.perspective(45, gl.viewportWidth / gl.viewportHeight, 0.1, 100.0,
        pMatrix);
    // 使用恒等矩阵进行初始化，将绘制位置设置在视图中央
    mat4.identity(mvMatrix);
    // 将绘制位置左移 1.5 个单位，内移 7 个单位（三角形第一个顶点位置处）
    mat4.translate(mvMatrix, [-1.5, 0.0, -7.0]);
    // 将当前的视图 - 模型矩阵保存在堆栈中
    mvPushMatrix();
    // 旋转三角形
    mat4.rotate(mvMatrix, degToRad(rTri), [0, 1, 0]);
    // 将 triangleVertexPositionBuffer 设定为接下来的操作所使用的缓冲区
    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexPositionBuffer);
    /* 通知 WebGL 上下文对象缓冲区中存放的顶点位置信息将作为三角形各顶点位置信息来使用，
       每个三角形使用三个数据（三维信息）*/
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
        triangleVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);
    // 将 triangleVertexColorBuffer 设定为接下来的操作所使用的缓冲区
    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexColorBuffer);
    /* 通知 WebGL 上下文对象缓冲区中存放的顶点颜色信息将作为三角形各顶点颜色信息来使用，
       每个三角形使用四个数据 */
    gl.vertexAttribPointer(shaderProgram.vertexColorAttribute,
        triangleVertexColorBuffer.itemSize, gl.FLOAT, false, 0, 0);

    // 通知 WebGL 上下文对象使用当前的模型 - 视图矩阵与投影矩阵
    setMatrixUniforms();
    // 绘制三角形
    gl.drawArrays(gl.TRIANGLES, 0, triangleVertexPositionBuffer.numItems);
    // 恢复使用堆栈中保存的视图 - 模型矩阵
    mvPopMatrix();
    // 将绘制位置右移 3 个单位
    mat4.translate(mvMatrix, [3.0, 0.0, 0.0]);
    // 将当前的视图 - 模型矩阵保存在堆栈中
    mvPushMatrix();
    // 旋转矩形
    mat4.rotate(mvMatrix, degToRad(rSquare), [1, 0, 0]);
    // 将 squareVertexPositionBuffer 设定为接下来的操作所使用的缓冲区
    gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexPositionBuffer);
    /* 通知 WebGL 缓冲区中存放的顶点位置信息将作为矩形各顶点位置信息来使用，
```

```

    每个矩形使用三个数据 (三维信息)*/
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
        squareVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);
    // 将 squareVertexColorBuffer 设定为接下来的操作所使用的缓冲区
    gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexColorBuffer);
    /* 通知 WebGL 缓冲区中存放的顶点颜色信息将作为矩形各顶点颜色信息来使用,
    每个矩形使用四个数据 */
    gl.vertexAttribPointer(shaderProgram.vertexColorAttribute,
        squareVertexColorBuffer.itemSize, gl.FLOAT, false, 0, 0);
    // 通知 WebGL 上下文对象使用当前的模型 - 视图矩阵与投影矩阵
    setMatrixUniforms();
    // 绘制矩形
    gl.drawArrays(gl.TRIANGLE_STRIP, 0,
        squareVertexPositionBuffer.numItems);
    // 恢复使用堆栈中保存的视图 - 模型矩阵
    mvPopMatrix();
}

```

注意该代码中 `gl.bindBuffer` 方法前的 `mvPushMatrix` 函数与 `mat4.rotate` 方法的使用, 以及 `gl.drawArrays` 方法后的 `mvPopMatrix` 函数的使用。前面介绍过, 在使用 WebGL 上下文对象进行 3D 图形的绘制时, 需要告诉 WebGL 上下文对象当前图形的绘制位置及旋转角度, 而当前图形的绘制位置及旋转角度都被保存在模型 - 视图矩阵中, 因此, `mat4.rotate` 方法的作用就比较明显了, 该方法的书写代码类似如下所示。

```
mat4.rotate(mvMatrix, degToRad(rTri), [0, 1, 0]);
```

其中 `mvMatrix` 代表视图 - 模型矩阵。该行代码的含义为将视图 - 模型矩阵中的旋转角度围绕垂直方向 (参数为 `[0,1,0]`) 旋转 `rTri` 度, 因为在 WebGL API 中, 使用弧度来指定旋转角度, 所以使用 `degToRad` 函数将角度转变成为弧度, 该函数代码如下所示。

```

// 将角度转变成弧度
function degToRad(degrees) {
    return degrees * Math.PI / 180;
}

```

那么, `mvPushMatrix` 函数与 `mvPopMatrix` 函数的作用又是什么呢? 当使用 `mat4.rotate` 方法旋转图形时, 实际上改变了视图 - 模型矩阵中保存的当前图形的旋转角度。例如在绘制三角形时, 当前图形的旋转角度在 X 轴、Y 轴以及 Z 轴上的旋转角度均为 0 度, 同时在绘制矩形时, 要求当前图形的旋转角度在 X 轴、Y 轴以及 Z 轴上的旋转角度仍然均为 0 度。但是由于使用 `mat4.rotate` 方法将三角形进行旋转, 因此改变了当前图形的旋转角度。在绘制矩形时, 当前图形的旋转角度已经改变, 这不是希望的结果, 所以需要在旋转三角形时先将当前的视图 - 模型矩阵保存在堆栈中, 旋转完毕后再将堆栈中保存的视图 - 模型矩阵恢复出来继续使用。`mvPushMatrix` 函数的作用是将当前的视图 - 模型矩阵保存在堆栈中, 函数代码如下所示。

```

var mvMatrix = mat4.create();
var mvMatrixStack = [];
var pMatrix = mat4.create();
// 将视图 - 模型矩阵保存在堆栈中
function mvPushMatrix()
{
    var copy = mat4.create();
    mat4.set(mvMatrix, copy);
    mvMatrixStack.push(copy);
}

```

而 mvPopMatrix 函数的作用是取出堆栈中保存的视图 - 模型矩阵，函数代码如下所示。

```

// 取出堆栈中保存的视图 - 模型矩阵
function mvPopMatrix()
{
    if (mvMatrixStack.length == 0)
        throw "Invalid popMatrix!";
    mvMatrix = mvMatrixStack.pop();
}

```

那么，为什么要将视图 - 模型矩阵保存在堆栈中呢？其意义在于利用任意数量的互相嵌套的绘制过程，可以将视图 - 模型矩阵一层层地向堆栈中保存，之后再一层层地从堆栈中恢复。

接下来回过头来看一下 tick 函数中第三行的 animate 函数。animate 函数的代码如下所示。

```

// 设置动画参数
function animate()
{
    // 获取当前时间
    var timeNow = new Date().getTime();
    // 如果动画绘制已经开始
    if (lastTime != 0)
    {
        // 获取当前时间与上次动画绘制时间的间隔时间
        var elapsed = timeNow - lastTime;
        // 计算三角形旋转角度，允许三角形每秒旋转 90 度
        rTri += (90 * elapsed) / 1000.0;
        // 计算矩形旋转角度，允许矩形每秒旋转 90 度
        rSquare += (75 * elapsed) / 1000.0;
    }
    // 保存动画绘制时间
    lastTime = timeNow;
}

```

很明显，在 animate 函数中，重新设置了每次重绘三角形与矩形时三角形与矩形的旋转角度。调用 animate 函数将三角形与矩形旋转一个固定的角度是一种使三角形与矩形旋转起来的非常简单的处理方法。一种更好的处理方法是根据现在时间与上一次重绘三角形与矩形的间隔时间来动态设置其旋转角度。这种方法允许三角形每秒旋转 90 度，矩形每秒旋转 75



度。这样处理的好处在于可以保证无论用户的计算机运转速度是快还是慢，用户看见的三角形与矩形的旋转速度都是相同的。在一个简单的示例程序中，这两种处理方法并没有什么太大的区别，但是在一个大型的网络游戏之中，第二种处理方法的优点就很明显了。

## 12.5 制作三维物体

本节将介绍如何使用 WebGL 上下文对象绘制一些有立体感的三维物体。

### 12.5.1 画面式样

本节将结合一个使用 WebGL 上下文对象制作三维物体的示例向读者介绍如何在 3D 视图使用 WebGL 上下文对象来绘制有立体感的三维物体。本示例程序以 12.4 节中的示例程序为基础，只不过本示例程序绘制的不是三角形与矩形，而是一个有立体感的椎体与立方体。本示例程序在浏览器中的显示效果如图 12-8 所示。

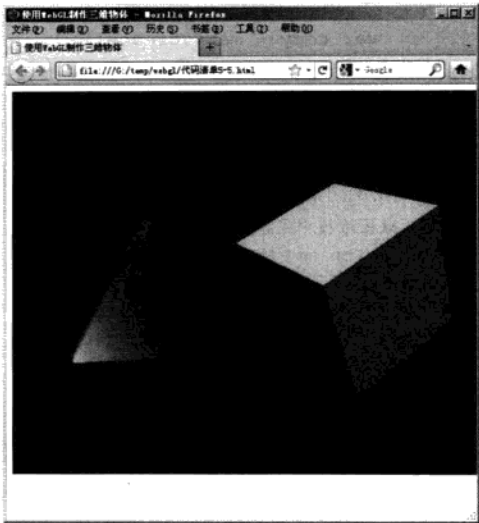


图 12-8 使用 WebGL 上下文对象绘制三维物体

### 12.5.2 代码剖析

本示例程序的代码以 12.4 节中的示例程序为基础进行修改，主要集中在对 `animate` 函数、`initBuffers` 函数以及 `drawScene` 函数的修改。首先来看一下 `animate` 函数中的代码，在该函数中，并不是使用变量 `rTri` 与变量 `rSquare` 来代表三角形与矩形的旋转角度，而是使用变量 `rPyramid` 与变量 `rCube` 来代表椎体与立方体的旋转角度。修改处的代码如下所示。

```
// 计算椎体旋转角度，允许椎体每秒旋转 90 度
rPyramid += (90 * elapsed) / 1000.0;
// 计算立方体旋转角度，允许立方体每秒旋转 90 度
rCube += (75 * elapsed) / 1000.0;
```

接下来看一下对于 `drawScene` 函数代码的修改。首先在该函数之前，将对变量 `rTri` 与变量 `rSquare` 的定义修改为对变量 `rPyramid` 与变量 `rCube` 的定义，代码如下所示。

```
var rPyramid = 0; // 椎体的旋转角度
var rCube = 0; // 立方体的旋转角度
```

接下来在 `drawScene` 函数中修改旋转三角形的代码为旋转椎体的代码，旋转矩形的代码为旋转立方体的代码，同时，在旋转立方体时，在 X 轴、Y 轴、Z 轴方向上同时旋转，代码如下所示。

```
// 旋转椎体
mat4.rotate(mvMatrix, degToRad(rPyramid), [0, 1, 0]);
// 旋转立方体
mat4.rotate(mvMatrix, degToRad(rSquare), [1, 1, 1]);
```

接下来将之前绘制三角形时使用的变量名修改为绘制椎体时使用的变量名，将绘制矩形时使用的变量名修改为绘制立方体时使用的变量名，代码如下所示。

```
// 将 pyramidVertexPositionBuffer 设定为接下来的操作所使用的缓冲区
gl.bindBuffer(gl.ARRAY_BUFFER, pyramidVertexPositionBuffer);
/* 通知 WebGL 上下文对象缓冲区中存放的顶点位置信息将作为椎体各顶点位置信息来使用，
每个椎体使用三个数据（三维信息）*/
gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
pyramidVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);
// 将 pyramidVertexColorBuffer 设定为接下来的操作所使用的缓冲区
gl.bindBuffer(gl.ARRAY_BUFFER, pyramidVertexColorBuffer);
/* 通知 WebGL 上下文对象缓冲区中存放的顶点颜色信息将作为椎体各顶点颜色信息来使用，
每个椎体使用四个数据 */
gl.vertexAttribPointer(shaderProgram.vertexColorAttribute,
pyramidVertexColorBuffer.itemSize, gl.FLOAT, false, 0, 0);
// 通知 WebGL 上下文对象使用当前的模型 - 视图矩阵与投影矩阵
setMatrixUniforms();
// 绘制椎体
gl.drawArrays(gl.TRIANGLES, 0, pyramidVertexPositionBuffer.numItems);
...
// 将 cubeVertexPositionBuffer 设定为接下来的操作所使用的缓冲区
gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
/* 通知 WebGL 上下文对象缓冲区中存放的顶点位置信息将作为立方体各顶点位置信息来使用，
每个立方体使用三个数据（三维信息）*/
gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
cubeVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);
// 将 cubeVertexColorBuffer 设定为接下来的操作时所使用的缓冲区
gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexColorBuffer);
/* 通知 WebGL 上下文对象缓冲区中存放的顶点颜色信息将作为立方体各顶点颜色信息来使用，
每个立方体使用四个数据 */
gl.vertexAttribPointer(shaderProgram.vertexColorAttribute,
cubeVertexColorBuffer.itemSize, gl.FLOAT, false, 0, 0);
```

在绘制立方体的时候，可以使用以下三个方法：

1) 依然使用一系列三角形进行绘制（参数为 `gl.TRIANGLE_STRIP`）。如果只使用同一种颜色来绘制立方体，推荐使用这种方法。可以使用之前使用的顶点位置信息创建立方体的第一个面，然后增加两个顶点来绘制第二个面，再增加两个顶点来绘制第三个面，使用同样的方法绘制立方体中的其他面，直到全部绘制完毕为止。这种方法是非常有效的。但是本示例想使立方体的每个面都具有不同的颜色，因为每个顶点指定了立方体的一个内角，每个内角由三个面共同拥有，所以每个顶点都需要指定三次，在这种情况下使用这种方法会非常不方便。

2) 可以通过绘制 6 个矩形的方法来绘制立方体（参数为 `gl.TRIANGLE_STRIP`），为每

个面绘制一个矩形，为每个矩形指定 4 个顶点及其颜色。但是这种方法不是最好的实现方法，因为，通知 WebGL 上下文对象绘制一个图形会花费大量时间（在这里需要为每个立方体绘制六个矩形），所以这种方法只适用绘制的图形量比较少的情況。

3) 将立方体定义为 6 个矩形，每个矩形由两个三角形组成，但是要求 WebGL 上下文对象一次将这 6 个矩形绘制完毕（参数为 `gl.TRIANGLE`）。这种方法有点类似与之前使用 `gl.TRIANGLE_STRIP` 参数绘制矩形时所使用的方法，但是现在每次都定义完整的三角形，而不是通过增加顶点的方法来定义新的三角形，所以可以很容易地定义三角形每条边的颜色。

现在，在绘制立方体中每个矩形中的三角形时，还有一些遗留问题。设想在绘制第一个面的时候，指定了 4 个顶点的位置，以及 4 个顶点的颜色。由于必须要使用两个三角形来进行绘制，但是代码中采用的是绘制普通三角形的方法（参数为 `gl.TRIANGLE`），因此这两个三角形各自需要独立的 3 个顶点与 3 个顶点的颜色，即总共需要指定 6 个顶点与 6 个顶点的颜色。但是在缓冲区中只想为立方体的每个面指定 4 个顶点与 4 个顶点的颜色，所以需要进行这样的指定：“使用缓冲区中的前 3 个顶点绘制一个三角形，使用第 1 个顶点，第 3 个顶点，第 4 个顶点绘制第 2 个三角形”，这样就可以绘制第 1 个面了，同时绘制立方体的其他面时也应该使用这个指定方法。

在这种情况下使用 WebGL API 中的一种新的类型的缓冲区——元素数组缓冲区（element array buffer）与一个新的方法——`drawElements` 方法，与之前使用的数组缓冲区（三角形与矩形所用的缓冲区）一样，需要在 `initBuffers` 函数中使用一个以 0 为索引基准点的顶点位置数组与顶点颜色数组对元素数组缓冲区进行填充。稍候将在对 `initBuffers` 函数的讲解中介绍这种元素数组缓冲区的填充方法。

为了使用这个元素数组缓冲区，需要将它指定为当前操作所用缓冲区，然后调用 `setMatrixUniforms` 方法通知 WebGL 上下文对象使用当前的模型 - 视图矩阵与投影矩阵，之后调用 `drawElement` 方法绘制立方体，代码如下所示。

```
// 将 cubeVertexIndexBuffer 设定为接下来的操作所使用的缓冲区
gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVertexIndexBuffer);
// 通知 WebGL 上下文对象使用当前的模型 - 视图矩阵与投影矩阵
setMatrixUniforms();
// 绘制立方体
gl.drawElements(gl.TRIANGLES, cubeVertexIndexBuffer.numItems,
gl.UNSIGNED_SHORT, 0);
```

最后来看一下 `drawScene` 函数中的完整代码，如下所示。

```
var rPyramid = 0; // 锥体的旋转角度
var rCube = 0; // 立方体的旋转角度
// 绘制图形
function drawScene()
{
    // 设置 3D 视图的视图大小
```

```

gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
// 擦除 canvas 元素中内容
gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
// 设置对视图的观察视角
mat4.perspective(45, gl.viewportWidth / gl.viewportHeight, 0.1, 100.0,
pMatrix);
// 使用恒等矩阵进行初始化, 将绘制位置设置在视图中央
mat4.identity(mvMatrix);
// 将绘制位置左移 1.5 个单位, 后移 7 个单位 (椎体第一个顶点位置处)
mat4.translate(mvMatrix, [-1.5, 0.0, -7.0]);
// 将当前的视图 - 模型矩阵保存在堆栈中
mvPushMatrix();
// 旋转椎体
mat4.rotate(mvMatrix, degToRad(rPyramid), [0, 1, 0]);
// 将 pyramidVertexPositionBuffer 设定为接下来的操作所使用的缓冲区
gl.bindBuffer(gl.ARRAY_BUFFER, pyramidVertexPositionBuffer);
/* 通知 WebGL 上下文对象缓冲区中存放的顶点位置信息将作为椎体各顶点位置信息来使用,
每个椎体使用三个数据 (三维信息) */
gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
pyramidVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);
// 将 pyramidVertexColorBuffer 设定为接下来的操作时所使用的缓冲区
gl.bindBuffer(gl.ARRAY_BUFFER, pyramidVertexColorBuffer);
/* 通知 WebGL 上下文对象缓冲区中存放的顶点颜色信息将作为椎体各顶点颜色信息来使用,
每个椎体使用四个数据 */
gl.vertexAttribPointer(shaderProgram.vertexColorAttribute,
pyramidVertexColorBuffer.itemSize, gl.FLOAT, false, 0, 0);

// 通知 WebGL 上下文对象使用当前的模型 - 视图矩阵与投影矩阵
setMatrixUniforms();
// 绘制椎体
gl.drawArrays(gl.TRIANGLES, 0, pyramidVertexPositionBuffer.numItems);
// 恢复使用堆栈中保存的视图 - 模型矩阵
mvPopMatrix();

// 将绘制位置右移 3 个单位
mat4.translate(mvMatrix, [3.0, 0.0, 0.0]);
// 将当前的视图 - 模型矩阵保存在堆栈中
mvPushMatrix();
// 旋转立方体
mat4.rotate(mvMatrix, degToRad(rCube), [1, 1, 1]);
// 将 cubeVertexPositionBuffer 设定为接下来的操作所使用的缓冲区
gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
/* 通知 WebGL 上下文对象缓冲区中存放的顶点位置信息将作为立方体各顶点位置信息来使用,
每个立方体使用三个数据 (三维信息) */
gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
cubeVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);
// 将 cubeVertexColorBuffer 设定为接下来的操作时所使用的缓冲区
gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexColorBuffer);
/* 通知 WebGL 上下文对象缓冲区中存放的顶点颜色信息将作为立方体各顶点颜色信息来使用,
每个立方体使用四个数据 */

```

```

gl.vertexAttribPointer(shaderProgram.vertexColorAttribute,
    cubeVertexColorBuffer.itemSize, gl.FLOAT, false, 0, 0);
// 将 cubeVertexIndexBuffer 设定为接下来的操作时所使用的缓冲区
gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVertexIndexBuffer);
// 通知 WebGL 上下文对象使用当前的模型 - 视图矩阵与投影矩阵
setMatrixUniforms();
// 绘制立方体
gl.drawElements(gl.TRIANGLES, cubeVertexIndexBuffer.numItems,
    gl.UNSIGNED_SHORT, 0);
// 恢复使用堆栈中保存的视图 - 模型矩阵
mvPopMatrix();
}

```

接下来看一下 `initBuffers` 函数中的代码。首先需要将之前代码中的三角形与矩形的顶点位置与顶点颜色缓冲区的变量名修改为椎体与立方体的顶点位置与顶点颜色缓冲区的变量名，同时添加代表元素数组缓冲区的变量名，如下所示。

```

var pyramidVertexPositionBuffer; // 椎体顶点位置缓冲区
var pyramidVertexColorBuffer; // 椎体各顶点颜色信息缓冲区
var cubeVertexPositionBuffer; // 立方体顶点位置缓冲区
var cubeVertexColorBuffer; // 立方体各顶点颜色信息缓冲区
var cubeVertexIndexBuffer; // 立方体元素数组缓冲区

```

接下来需要在椎体顶点位置缓冲区中指定各个面的顶点信息，并且为 `numItems` 属性指定一个合适的属性值（椎体的顶点个数），代码如下所示。

```

// 初始化缓冲区
function initBuffers()
{
    // 创建椎体顶点位置缓冲区
    pyramidVertexPositionBuffer = gl.createBuffer();
    // 将 pyramidVertexPositionBuffer 设定为接下来的操作所使用的缓冲区
    gl.bindBuffer(gl.ARRAY_BUFFER, pyramidVertexPositionBuffer);
    // 使用 JavaScript 列表定义一个椎体的一组顶点信息
    var vertices = [
        // 前面
        0.0, 1.0, 0.0,
        -1.0, -1.0, 1.0,
        1.0, -1.0, 1.0,
        // 右面
        0.0, 1.0, 0.0,
        1.0, -1.0, 1.0,
        1.0, -1.0, -1.0,
        // 后面
        0.0, 1.0, 0.0,
        1.0, -1.0, -1.0,
        -1.0, -1.0, -1.0,
        // 左面
        0.0, 1.0, 0.0,

```

```

        -1.0, -1.0, -1.0,
        -1.0, -1.0,  1.0
    ];
    // 使用顶点列表创建 Float32Array 对象填充缓存
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices),
    gl.STATIC_DRAW);
    // 缓冲区中存放 12 个顶点信息, 每个顶点由三个数据(三维)构成
    pyramidVertexPositionBuffer.itemSize = 3;
    pyramidVertexPositionBuffer.numItems = 12;

```

接下来采用类似方法指定椎体各顶点颜色信息的缓冲区, 代码如下所示。

```

// 创建椎体各顶点颜色信息的缓冲区
pyramidVertexColorBuffer = gl.createBuffer();
// 将该缓冲区指定为当前操作所用的缓冲区
gl.bindBuffer(gl.ARRAY_BUFFER, pyramidVertexColorBuffer);
// 使用 JavaScript 列表定义椎体各顶点所用颜色
var colors = [
    // 前面
    1.0, 0.0, 0.0, 1.0,
    0.0, 1.0, 0.0, 1.0,
    0.0, 0.0, 1.0, 1.0,
    // 右面
    1.0, 0.0, 0.0, 1.0,
    0.0, 0.0, 1.0, 1.0,
    0.0, 1.0, 0.0, 1.0,
    // 后面
    1.0, 0.0, 0.0, 1.0,
    0.0, 1.0, 0.0, 1.0,
    0.0, 0.0, 1.0, 1.0,
    // 左面
    1.0, 0.0, 0.0, 1.0,
    0.0, 0.0, 1.0, 1.0,
    0.0, 1.0, 0.0, 1.0
];
// 使用顶点颜色列表创建 Float32Array 对象填充缓存
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(colors),
gl.STATIC_DRAW);
// 缓冲区中存放 12 个顶点信息, 每个顶点由 4 个数据构成
pyramidVertexColorBuffer.itemSize = 4;
pyramidVertexColorBuffer.numItems = 12;

```

接下来在立方体顶点位置缓冲区中指定各个面的顶点信息, 代码如下所示。

```

// 创建立方体顶点位置缓冲区
cubeVertexPositionBuffer = gl.createBuffer();
// 将 cubeVertexPositionBuffer 设定为接下来的操作所使用的缓冲区
gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
// 利用 JavaScript 列表定义一个立方体的一组顶点信息
vertices = [
    // 前面

```

```

-1.0, -1.0, 1.0,
 1.0, -1.0, 1.0,
 1.0, 1.0, 1.0,
-1.0, 1.0, 1.0,

// 后面
-1.0, -1.0, -1.0,
-1.0, 1.0, -1.0,
 1.0, 1.0, -1.0,
-1.0, -1.0, -1.0,

// 顶面
-1.0, 1.0, -1.0,
-1.0, 1.0, 1.0,
 1.0, 1.0, 1.0,
 1.0, 1.0, -1.0,

// 底面
-1.0, -1.0, -1.0,
 1.0, -1.0, -1.0,
 1.0, -1.0, 1.0,
-1.0, -1.0, 1.0,

// 右面
 1.0, -1.0, -1.0,
 1.0, 1.0, -1.0,
 1.0, 1.0, 1.0,
 1.0, -1.0, 1.0,

// 左面
-1.0, -1.0, -1.0,
-1.0, -1.0, 1.0,
-1.0, 1.0, 1.0,
-1.0, 1.0, -1.0,
];
// 使用顶点列表创建 Float32Array 对象填充缓存
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices),
gl.STATIC_DRAW);
// 缓冲区中存放 24 个顶点信息, 每个顶点由 3 个数据 (三维) 构成
cubeVertexPositionBuffer.itemSize = 3;
cubeVertexPositionBuffer.numItems = 24;

```

指定立方体各顶点颜色信息的缓冲区的代码稍微复杂一点, 因为不想为每个面中的 4 个顶点书写 4 次指定同样颜色的代码, 所以利用循环来创建一个顶点颜色信息的列表, 代码如下所示。

```

// 创建立方体各顶点颜色信息的缓冲区
cubeVertexColorBuffer = gl.createBuffer();
// 将该缓冲区指定为当前操作所用缓冲区
gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexColorBuffer);

```

```

// 使用 JavaScript 列表定义立方体各顶点所用颜色
colors = [
    [1.0, 0.0, 0.0, 1.0], // 前面
    [1.0, 1.0, 0.0, 1.0], // 后面
    [0.0, 1.0, 0.0, 1.0], // 顶面
    [1.0, 0.5, 0.5, 1.0], // 底面
    [1.0, 0.0, 1.0, 1.0], // 右面
    [0.0, 0.0, 1.0, 1.0], // 左面
];
var unpackedColors = [];
for (var i in colors) {
    var color = colors[i];
    for (var j=0; j < 4; j++) {
        unpackedColors = unpackedColors.concat(color);
    }
}
// 利用顶点颜色列表创建 Float32Array 对象填充缓存
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(unpackedColors),
gl.STATIC_DRAW);
// 缓冲区中存放 24 个顶点信息, 每个顶点由 4 个数据构成
cubeVertexColorBuffer.itemSize = 4;
cubeVertexColorBuffer.numItems = 24;

```

最后定义元素数组缓冲区, 代码如下所示 (注意 `gl.bindBuffer` 方法与 `gl.bufferData` 方法中第一个参数使用的是 `gl.ELEMENT_ARRAY_BUFFER`)。

```

// 创建立方体所用的元素数组缓冲区
cubeVertexIndexBuffer = gl.createBuffer();
// 将该缓冲区指定为当前操作所用缓冲区
gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVertexIndexBuffer);
// 指定每个面的两个三角形中应该使用顶点的序号
var cubeVertexIndices = [
    0, 1, 2,      0, 2, 3,      // 前面
    4, 5, 6,      4, 6, 7,      // 后面
    8, 9, 10,      8, 10, 11,    // 顶面
    12, 13, 14,     12, 14, 15,   // 底面
    16, 17, 18,     16, 18, 19,   // 右面
    20, 21, 22,     20, 22, 23    // 左面
];
// 使用列表创建 Uint16Array 对象填充缓存
gl.bufferData(gl.ELEMENT_ARRAY_BUFFER,
new Uint16Array(cubeVertexIndices), gl.STATIC_DRAW);
// 缓冲区中存放 36 个数据信息
cubeVertexIndexBuffer.itemSize = 1;
cubeVertexIndexBuffer.numItems = 36;
}

```

记住, 这个缓冲区中的每一个数字代表顶点位置缓冲区与顶点颜色缓冲区中某一个顶点的序号, 所以第一行的 6 个数字表示在绘制立方体的前面时, 使用第 0 个、第 1 个和第 2 个



顶点来定义第一个三角形,使用第0个、第2个顶点和第3个顶点来定义第2个三角形,在绘制前面的矩形的时候,使用第0个、第1个、第2个和第3个顶点来定义第一个矩形,绘制其他面的矩形的时候也使用这个方法。

## 12.6 使用纹理

本节介绍如何对3D物体绘制纹理。从一个文件中读取图像,然后将该图像作为纹理绘制在已经绘制好的3D物体上。在为某个3D物体绘制纹理时这是一种非常有用的方法,因为这样可以避免将物体绘制得非常复杂。比如想在某个3D游戏中绘制一堵石头墙,不需要为每一个砖块绘制一个物体,只需绘制一块砖的图像,然后将它作为纹理绘制在石头墙上就可以了。

### 12.6.1 画面式样

本示例程序在12.5节示例程序的基础上加以修改,删除其中对椎体的绘制,只绘制立方体,然后对立方体绘制纹理。本示例程序在浏览器中的显示效果如图12-9所示。

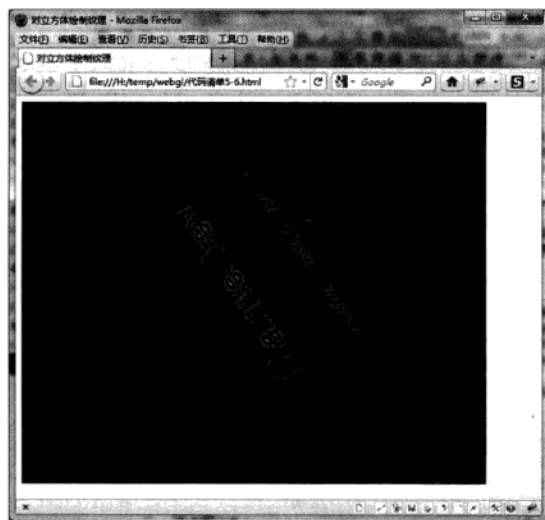


图 12-9 对立方体绘制纹理

### 12.6.2 代码剖析

首先需要说明的是,对三维物体绘制纹理,实际上是一种特殊的对三维物体使用颜色的操作。在12.3节中介绍过,图形或物体的颜色是通过片元渲染器来绘制的,因此需要装载图像并将它传入片元渲染器中;同时还要通知片元渲染器在为图形或图像绘制每一个像素时需要使用图像中相对应的哪一个像素来进行绘制,因此也需要向片元渲染器中传入这个信息。

接着看一下有关装载图像的代码。首先,在webGLStart函数中添加initTexture()函数,

webGLStart 函数的完整代码如下所示。

```
// 绘制 3D 图形
function webGLStart()
{
    var canvas = document.getElementById("canvas1");// 获取 canvas 元素
    initGL(canvas);// 初始化 WebGL 上下文对象
    initShaders();// 初始化渲染器
    initBuffers();// 初始化缓冲区
    initTexture();// 初始化纹理
    // 每次清除 canvas 元素中的内容时均将其填充为黑色
    gl.clearColor(0.0, 0.0, 0.0, 1.0);
    gl.enable(gl.DEPTH_TEST);// 开启深度测试
    tick();// 调用 tick 函数
}
```

接下来看一下 initTexture 函数中的代码，如下所示。

```
var myTexture;// 纹理对象
// 初始化纹理
function initTexture()
{
    myTexture = gl.createTexture();// 创建纹理对象
    // 创建 Image 对象并将纹理对象的 image 属性值设定为该 Image 对象
    myTexture.image = new Image();
    myTexture.image.onload = function () {
        // 调用 handleLoadedTexture 函数
        handleLoadedTexture(myTexture)
    }

    // 指定纹理图像来源
    myTexture.image.src = "texture.gif";
}
```

在这段代码中，先创建了一个代表纹理对象的全局变量，接下来使用 gl.createTexture 方法创建一个纹理对象，之后创建一个 JavaScript 的 Image 对象并将纹理对象的 image 属性的属性值设定为该 Image 对象。很明显下一步工作是装载实际图像。这里在代码中指定了一个 handleLoadedTexture 函数，每次图像装载完毕之后都会调用这个函数。最后，利用 Image 对象的 src 属性来指定纹理对象所用的图片来源。图像将会以异步的方式被装载，在 Web 服务器中将会使用一个后台线程来装载图像。图像装载完毕后，将会调用 handleLoadedTexture 函数，该函数的代码如下所示。

```
// 处理装载完毕的纹理
function handleLoadedTexture(texture)
{
    // 通知 WebGL 上下文对象 texture 为当前所使用的纹理
    gl.bindTexture(gl.TEXTURE_2D, texture);
    // 通知 WebGL 上下文对象将纹理图像垂直翻转
    gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE,
```

```

texture.image); // 将纹理图像传入显卡
// 指定纹理过滤的参数
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST);
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.NEAREST);
}

```

在该函数中，先将 WebGL 绘制图形时所使用的当前纹理设定为函数的传入参数 texture 所代表的纹理，这个概念有点类似于在 initBuffers 函数中将某个缓冲区指定为当前所使用的缓冲区。

接下来需要通知 WebGL 在装载纹理图像时先将其在垂直方向上进行翻转。这是因为从 WebGL API 中的投影这个角度上来看，使用的 GIF 格式的图像实际上是已经被垂直翻转过了的，所以需要在程序中将它再次垂直翻转回来。

接下来需要使用 `textureImage2D` 方法将装载完毕的纹理图像传入显卡之中。该方法中的参数依次为：使用的图像类型（`gl.TEXTURE_2D` 表示该图像是一个 2D 图像）、图像的详细程度（一般设为 0 即可）、以什么格式存储在显卡中（RGBA 代表以 RGBA 格式存储，其中 R 表示红色值、G 表示绿色值、B 表示蓝色值、A 表示透明度值，重复两次的原因将在后面进行介绍），`gl.UNSIGNED_BYTE` 表示组成图像的数据是无符号字节类型，`texture.image` 代表纹理图像（JavaScript 中的 Image 对象）。

在接下来的两行中为纹理图像指定了两个与缩放有关的参数。这里首先介绍 WebGL 中的纹理过滤概念。当三维空间中的图形或物体经过坐标变换和投影等过程，变成二维屏幕上的一组像素的时候，每个像素需要到相应的纹理图像中进行采样，这个过程就称为纹理过滤。纹理过滤通常分为 2 种情况：1) 纹理被缩小。比如将一个  $8 \times 8$  的纹理贴到一个平行于 xy 平面的正方形上，最后该正方形在屏幕上只占  $4 \times 4$  的像素矩阵，在这种情况下一个像素对应着多个纹理单元。2) 纹理被放大。这种情况刚好与上一种情况相反，假如放大该正方形，最后正方形在屏幕上占了一个  $16 \times 16$  的像素矩阵，这样就变成一个纹理单元对应多个像素。在这两行代码中，参数 `gl.TEXTURE_MAG_FILTER` 的作用是对纹理的放大过滤指定参数，而参数 `gl.TEXTURE_MIN_FILTER` 的作用是对纹理的缩小过滤指定参数，将纹理放大（参数为 `gl.TEXTURE_MAG_FILTER`）与纹理缩小（参数为 `gl.TEXTURE_MIN_FILTER`）时指定的纹理方式参数值都设定为 `gl.NEAREST` 表示使用最近点采样方式，在对图像中某个像素进行绘制时会在纹理图像中寻找与该像素最接近的一个像素点上的颜色进行绘制。绘制结果为放大图形或物体的局部特写时图像会一块一块地不平滑显示，但是绘制起来比较快速，在绘制的性能上比较有优势。在下一节中将指定另外一个参数，以比较两种参数在图像的显示与性能方面的优缺点。

接下来看一下 `initBuffers` 函数中的代码。在该函数中，先删除 12.5 节示例程序中绘制锥体的有关代码。同时，将原程序中使用立方体顶点颜色缓冲区的代码修改为使用立方体顶点纹理缓冲区的有关代码。`initBuffers` 函数中的完整代码如下所示。

```

var cubeVertexPositionBuffer; // 立方体顶点位置缓冲区
var cubeVertexTextureCoordBuffer; // 立方体各顶点所用纹理缓冲区

```

```

var cubeVertexIndexBuffer;// 立方体元素数组缓冲区
// 初始化缓冲区
function initBuffers()
{
    // 创建立方体顶点位置缓冲区
    cubeVertexPositionBuffer = gl.createBuffer();
    // 将 cubeVertexPositionBuffer 设定为接下来的操作所使用的缓冲区
    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
    // 使用 JavaScript 列表定义一个立方体的一组顶点信息
    vertices = [
        // 前面
        -1.0, -1.0,  1.0,
        1.0, -1.0,  1.0,
        1.0,  1.0,  1.0,
        -1.0,  1.0,  1.0,

        // 后面
        -1.0, -1.0, -1.0,
        -1.0,  1.0, -1.0,
        1.0,  1.0, -1.0,
        1.0, -1.0, -1.0,

        // 顶面
        -1.0,  1.0, -1.0,
        -1.0,  1.0,  1.0,
        1.0,  1.0,  1.0,
        1.0,  1.0, -1.0,

        // 底面
        -1.0, -1.0, -1.0,
        1.0, -1.0, -1.0,
        1.0, -1.0,  1.0,
        -1.0, -1.0,  1.0,

        // 右面
        1.0, -1.0, -1.0,
        1.0,  1.0, -1.0,
        1.0,  1.0,  1.0,
        1.0, -1.0,  1.0,

        // 左面
        -1.0, -1.0, -1.0,
        -1.0, -1.0,  1.0,
        -1.0,  1.0,  1.0,
        -1.0,  1.0, -1.0,
    ];
    // 使用顶点列表创建 Float32Array 对象填充缓存
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices),
    gl.STATIC_DRAW);
    // 缓冲区中存放 24 个顶点信息, 每个顶点由三个数据 (三维) 构成

```

```

cubeVertexPositionBuffer.itemSize = 3;
cubeVertexPositionBuffer.numItems = 24;

// 建立立方体各顶点所用的纹理缓冲区
cubeVertexTextureCoordBuffer = gl.createBuffer();
// 将 cubeVertexTextureCoordBuffer 设定为接下来的操作所使用的缓冲区
gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexTextureCoordBuffer);
// 使用 JavaScript 列表定义一个立方体的一组顶点所用的纹理信息
var textureCoords = [
    // 前面
    0.0, 0.0,
    1.0, 0.0,
    1.0, 1.0,
    0.0, 1.0,

    // 后面
    1.0, 0.0,
    1.0, 1.0,
    0.0, 1.0,
    0.0, 0.0,

    // 顶面
    0.0, 1.0,
    0.0, 0.0,
    1.0, 0.0,
    1.0, 1.0,

    // 底面
    1.0, 1.0,
    0.0, 1.0,
    0.0, 0.0,
    1.0, 0.0,

    // 右面
    1.0, 0.0,
    1.0, 1.0,
    0.0, 1.0,
    0.0, 0.0,

    // 左面
    0.0, 0.0,
    1.0, 0.0,
    1.0, 1.0,
    0.0, 1.0,
];
// 使用顶点纹理列表创建 Float32Array 对象填充缓存
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(textureCoords),
gl.STATIC_DRAW);
// 缓冲区中存放 24 个顶点信息, 每个顶点由两个数据构成
cubeVertexTextureCoordBuffer.itemSize = 2;

```

```

cubeVertexTextureCoordBuffer.numItems = 24;

// 建立立方体所用的元素数组缓冲区
cubeVertexIndexBuffer = gl.createBuffer();
// 将该缓冲区指定为当前操作时所用的缓冲区
gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVertexIndexBuffer);
// 指定每个面的两个三角形中应该使用的顶点的序号
var cubeVertexIndices = [
    0, 1, 2,      0, 2, 3,      // 前面
    4, 5, 6,      4, 6, 7,      // 后面
    8, 9, 10,     8, 10, 11,     // 顶面
    12, 13, 14,   12, 14, 15,    // 底面
    16, 17, 18,   16, 18, 19,    // 右面
    20, 21, 22,   20, 22, 23,    // 左面
]
// 利用列表创建 Uint16Array 对象填充缓存
gl.bufferData(gl.ELEMENT_ARRAY_BUFFER,
new Uint16Array(cubeVertexIndices), gl.STATIC_DRAW);
// 缓冲区中存放 36 个数据信息
cubeVertexIndexBuffer.itemSize = 1;
cubeVertexIndexBuffer.numItems = 36;
}

```

这段代码创建了一个新的纹理缓冲区，在该缓冲区为每个立方体的顶点指定两个值，分别为纹理坐标系中的 X 轴上的值与 Y 轴上的值。在纹理坐标系中，将整个纹理图像定义为 1.0（宽）×1.0（高），所以 (0,0) 代表纹理图像的左下角，(1,1) 代表了纹理图像的右上角，而从这个分辨率转换到绘制时真实的纹理图像的分辨率是靠 WebGL 自动完成的。

接下来看一下 drawScene 函数中的代码。在该函数中，值得注意的是对纹理的使用。另外，在该函数中，删除了在 12.5 节示例程序中绘制椎体的有关代码，同时，修改了立方体的旋转方式。drawScene 函数中的完整代码如下所示。

```

var xRot = 0;
var yRot = 0;
var zRot = 0;
// 绘制图形
function drawScene()
{
    // 设置 3D 视图的视图大小
    gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
    // 擦除 canvas 元素中内容
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
    // 设置对视图的观察视角
    mat4.perspective(45, gl.viewportWidth / gl.viewportHeight, 0.1, 100.0,
    pMatrix);
    // 使用恒等矩阵进行初始化，将绘制位置设置在视图中央
    mat4.identity(mvMatrix);

    // 将绘制位置后移 5 个单位

```

```

mat4.translate(mvMatrix, [0.0, 0.0, -5.0]);
// 旋转立方体
mat4.rotate(mvMatrix, degToRad(xRot), [1, 0, 0]);
mat4.rotate(mvMatrix, degToRad(yRot), [0, 1, 0]);
mat4.rotate(mvMatrix, degToRad(zRot), [0, 0, 1]);
// 将 cubeVertexPositionBuffer 设定为接下来的操作所使用的缓冲区
gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
/* 通知 WebGL 缓冲区中存放的顶点位置信息将作为立方体各顶点位置信息来使用,
每个立方体使用三个数据 (三维信息) */
gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
cubeVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);
// 将 cubeVertexTextureCoordBuffer 设定为接下来的操作所使用的缓冲区
gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexTextureCoordBuffer);
// 通知 WebGL 上下文对象缓冲区中存放的顶点纹理信息将作为立方体各顶点纹理信息来使用
gl.vertexAttribPointer(shaderProgram.textureCoordAttribute,
cubeVertexTextureCoordBuffer.itemSize, gl.FLOAT, false, 0, 0);
// 通知 WebGL 上下文对象最先处理的纹理为 TEXTURE0
gl.activeTexture(gl.TEXTURE0);
gl.bindTexture(gl.TEXTURE_2D, myTexture);
// 设置程序对象中的一致变量的值为 0
gl.uniform1i(shaderProgram.samplerUniform, 0);
// 将 cubeVertexIndexBuffer 设定为接下来的操作所使用的缓冲区
gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVertexIndexBuffer);
// 通知 WebGL 上下文对象使用当前的模型 - 视图矩阵与投影矩阵
setMatrixUniforms();
// 绘制立方体
gl.drawElements(gl.TRIANGLES, cubeVertexIndexBuffer.numItems,
gl.UNSIGNED_SHORT,
0);
// 恢复使用堆栈中保存的视图 - 模型矩阵
mvPopMatrix();
}

```

注意该函数代码中如下三行代码。

```

gl.activeTexture(gl.TEXTURE0);
gl.bindTexture(gl.TEXTURE_2D, myTexture);
gl.uniform1i(shaderProgram.samplerUniform, 0);

```

在 WebGL API 中, 当使用 `gl.drawElements` 方法绘制物体时, 最多可以处理 32 个纹理对象, 分别使用 `TEXTURE0` 到 `TEXTURE31` 来代表这 32 个纹理对象。这三行中的第一行表示最先处理的纹理对象为 `TEXTURE0`。第三行代码的含义为将当前所使用的纹理对象的序号 0 传入程序对象中的一个一致变量 `uSampler` 中, 稍后将介绍如何使用这个一致变量 `uSampler`。

接下来看一下渲染器中的代码, 首先是顶点渲染器中的代码, 如下所示。

```

<script id="shader-vs" type="x-shader/x-vertex">
attribute vec3 aVertexPosition;
attribute vec2 aTextureCoord;

```

```

uniform mat4 uMVMatrix;
uniform mat4 uPMatrix;
varying vec2 vTextureCoord;
void main(void)
{
    gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
    vTextureCoord = aTextureCoord;
}
</script>

```

这段代码与前面示例程序中的顶点渲染器中的代码非常类似，唯一区别是为每一个顶点接收它在纹理坐标系中的坐标并传入到一个易变变量中。

为每一个顶点调用了这段渲染器代码之后，WebGL 将会计算出每一个顶点所封闭的区域中每一个片元（默认为像素）在使用了线性插值处理后的颜色值。

接下来看一下片元渲染器中的代码，如下所示。

```

<script id="shader-fs" type="x-shader/x-fragment">
#ifdef GL_ES
precision highp float;
#endif
varying vec2 vTextureCoord;
uniform sampler2D uSampler;
void main(void)
{
    gl_FragColor = texture2D(uSampler, vec2(vTextureCoord.s,
        vTextureCoord.t));
}
</script>

```

在这段代码中，所有顶点的片元渲染器都会使用 `texture2D` 方法从纹理坐标系中获取该顶点在纹理图像中的颜色。在纹理坐标系中，不是使用 `x,y` 来代表水平坐标点与垂直坐标点，而是使用 `s,t` 来代表水平坐标点与垂直坐标点，但是渲染器语言支持别名，所以依然可以使用 `vTextureCoord.x` 和 `vTextureCoord.y`。

接下来看一下对 `initShaders` 函数中的修改。在该函数中，删除对 `aVertexColor` 属性的引用，添加对 `aTextureCoord` 属性的引用并将它保存到程序对象的 `textureCoord` 属性中（在 `drawScene` 函数中，将程序对象的 `textureCoord` 属性值设定为缓冲区中各顶点在纹理坐标系中的坐标信息，通过此处代码将 `textureCoord` 属性的属性值设定为 `aTextureCoord` 属性的属性值；在顶点渲染器中，将属性值设定为易变变量 `vTextureCoord` 的属性值；在片元渲染器中，将该属性值作为各顶点在纹理坐标系中的坐标信息进行绘制）。同时获取一致变量 `uSampler` 的定位信息，并将它保存在程序对象的 `samplerUniform` 属性中（因此由于在 `drawScene` 函数中将程序对象的 `samplerUniform` 属性值设为 0 时，一致变量 `uSampler` 也被设为 0，因此在调用片元渲染器进行处理的时候，使用的是 `TEXTURE0` 纹理对象）。



initShaders 函数中的完整代码如下所示。

```
var shaderProgram;// 程序对象
// 初始化渲染器
function initShaders()
{
    // 获取片元渲染器
    var fragmentShader = getShader(gl, "shader-fs");
    // 获取顶点渲染器
    var vertexShader = getShader(gl, "shader-vs");
    // 创建程序对象
    shaderProgram = gl.createProgram();
    // 将渲染器绑定到程序对象上
    gl.attachShader(shaderProgram, vertexShader);
    gl.attachShader(shaderProgram, fragmentShader);
    gl.linkProgram(shaderProgram);
    // 如果不能初始化渲染器
    if (!gl.getProgramParameter(shaderProgram, gl.LINK_STATUS))
        alert(" 不能初始化渲染器 ");

    gl.useProgram(shaderProgram);
    /* 获取程序对象的 aVertexPosition 属性的引用并将其保存在程序对象的
    vertexPosition 属性中 */
    shaderProgram.vertexPositionAttribute =
    gl.getAttribLocation(shaderProgram, "aVertexPosition");
    // 使用数组来提供 vertexPositionAttribute 属性的值
    gl.enableVertexAttribArray(shaderProgram.vertexPositionAttribute);
    /* 获取程序对象的 aTextureCoord 属性的引用并将其保存在程序对象的
    textureCoord 属性中 */
    shaderProgram.textureCoordAttribute =
    gl.getAttribLocation(shaderProgram, "aTextureCoord");
    // 利用数组来提供 textureCoord 属性的值
    gl.enableVertexAttribArray(shaderProgram.textureCoordAttribute);

    shaderProgram.pMatrixUniform = gl.getUniformLocation(shaderProgram,
    "uPMatrix");
    shaderProgram.mvMatrixUniform = gl.getUniformLocation(shaderProgram,
    "uMVMMatrix");
    shaderProgram.samplerUniform = gl.getUniformLocation(shaderProgram,
    "uSampler");
}
```

最后来看一下 animate 函数的完整代码。在该函数中使用 xRot、yRot 及 zRot 变量来设置旋转角度，同时修改了立方体的旋转方式，代码如下所示。

```
// 设置动画参数
function animate()
{
    // 获取当前时间
    var timeNow = new Date().getTime();
```

```
// 如果动画绘制已经开始
if (lastTime != 0)
{
    // 获取当前时间与上次动画绘制时间的间隔时间
    var elapsed = timeNow - lastTime;
    xRot += (90 * elapsed) / 1000.0;
    yRot += (90 * elapsed) / 1000.0;
    zRot += (90 * elapsed) / 1000.0;
}
lastTime = timeNow;
}
```

## 12.7 键盘输入与纹理过滤

本节将介绍如何让用户通过键盘输入来对利用 WebGL 上下文对象制作出来的三维图像进行控制。允许用户通过按键来改变一个拥有纹理的立方体的旋转速度与旋转方向，同时允许用户指定是使用快速但质量低的纹理过滤方法，还是使用慢速但质量高的纹理过滤方法。

### 12.7.1 画面式样

本示例程序是由 12.6 节的示例程序修改而来的。在页面中添加了几行说明文字，同时允许用户通过按键方式来改变立方体旋转方向、旋转速度及纹理过滤方法。

本示例程序在浏览器中的显示效果如图 12-10 所示。

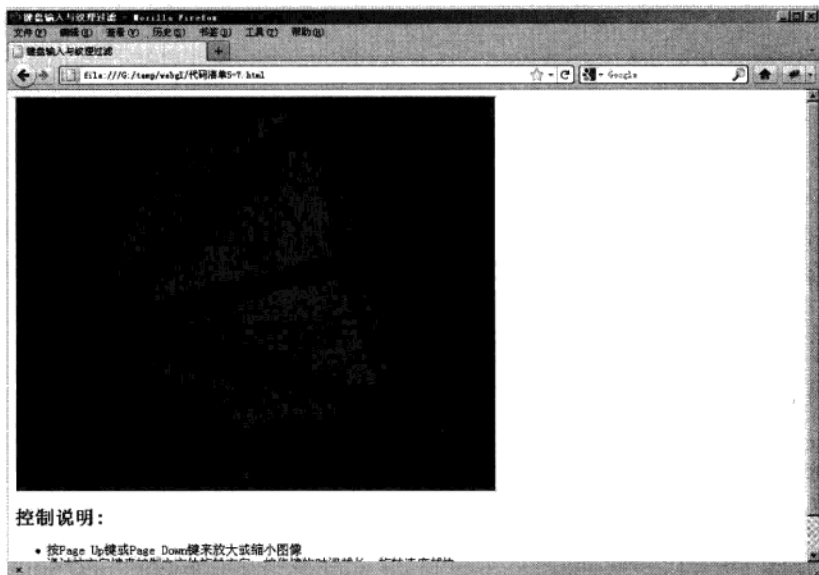


图 12-10 通过按键来改变旋转速度、旋转方向及纹理过滤方式的显示效果

## 12.7.2 代码剖析

### 1. HTML 页面代码

接下来看一下该示例程序的 HTML 页面代码部分，该页面在 12.6 节的示例页面的基础上添加了几行说明文字，代码如代码清单 12-3 所示。

代码清单 12-3 示例程序的 HTML 页面代码

---

```
<html>
<head>
<title> 键盘输入与纹理过滤 </title>
<meta http-equiv="content-type" content="text/html; charset=utf-8">
<script type="text/javascript" src="glMatrix-0.9.5.min.js"></script>
... (JavaScript 脚本代码稍后介绍) ...
</head>
<body onload="webGLStart();">
<canvas id="canvas1" style="border: none;" width="600"
height="500"></canvas>
<h2> 控制说明 :</h2>
<ul>
    <li> 按 <code>Page Up</code> 键或 <code>Page Down</code> 键来放大或缩小图像
    <li> 通过按方向键来控制立方体旋转方向，按住键的时间越长，旋转速度越快
    <li> 通过按 <code>F</code> 键来改变纹理过滤方法
</ul>
</body>
</html>
```

---

### 2. 脚本代码

接下来看一下该示例程序的脚本代码部分。

本示例程序与 12.6 节的示例程序的最大区别是，本示例程序中的三维图像可以接收用户的键盘输入事件。接收用户的键盘输入事件的实现原理非常简单，现在具体讲解一下。

首先，修改 12.6 节示例程序中关于变量 `xRot`（代表 X 轴上的旋转角度）、变量 `yRot`（代表 Y 轴上的旋转角度）和变量 `zRot`（代表 Z 轴上的旋转角度）的代码，删除变量 `zRot`，添加变量 `z`（代表绘制立方体时在 Z 轴坐标上的绘制位置），添加变量 `xSpeed`（代表立方体的横向旋转速度）、变量 `ySpeed`（代表立方体的纵向旋转速度）以及变量 `filter`（决定纹理过滤方式）。代码如下所示。

```
var xRot = 0; // X 轴方向上的旋转角度
var xSpeed = 0; // 横向旋转速度
var yRot = 0; // Y 轴方向上的旋转角度
var ySpeed = 0; // 纵向旋转速度
var z = -5.0; // Z 轴上的绘制位置
var filter = 0; // 纹理过滤方式
```

本示例程序将允许用户通过按下 `PageUp` 键或 `PageDown` 键来改变立方体在 Z 轴上的绘

制位置，或者说将立方体拉近或拉远，最终用户将看到拉近立方体时立方体会变得越来越大，拉远立方体时立方体会变得越来越小。

接下来，在介绍 `handleLoadedTexture` 函数之前，先来看一下在 WebGL 中使用的几种纹理过滤方式。

### 1) 最近点采样方式。

在 12.6 节中，使用过这种纹理过滤方式。使用这种纹理过滤方式时需要将纹理放大（参数为 `gl.TEXTURE_MAG_FILTER`）和纹理缩小（参数为 `gl.TEXTURE_MIN_FILTER`）时的纹理过滤方式参数值都设定为 `gl.NEAREST`。这种采样方式的含义为：在对图像中某个像素进行绘制时会在纹理图像中寻找与该像素最接近的一个像素点上的颜色进行绘制。其绘制结果为放大图形或物体的局部特写时图像会一块一块地不平滑显示，但是绘制起来速度比较快。

### 2) 线性过滤方式。

使用这种纹理过滤方式时需要将纹理放大和纹理缩小时的纹理过滤方式参数值都设定为 `gl.LINEAR`。在将纹理进行放大时这种线性过滤方式绘制出来的图像会更平滑一些，因为它在对图像中某个像素进行绘制时会在纹理图像中寻找与该像素最接近的  $2 \times 2$  个像素矩阵中的 4 个像素点上的颜色的平均值进行绘制。这种纹理过滤方式的缺点是绘制速度要比最近点采样方式的绘制速度慢很多。

### 3) 三线性过滤方式。

使用这种纹理过滤方式时需要将纹理放大时的纹理过滤方式参数值设定为 `gl.LINEAR`，将纹理缩小时的纹理过滤方式参数值设定为 `gl.LINEAR_MIPMAP_NEAREST`。这是三种纹理过滤方式中最复杂的一种过滤方式。该方式在纹理放大时依然使用线性过滤方式，但在纹理缩小时，使用线性过滤方式绘制出来的图像是非常不平滑，非常模糊的，而采用三线性过滤方式可以很好地解决这个问题。在纹理缩小时，先构造纹理图像的 `mipmap`。`mipmap` 是对最初的纹理图像构造的一系列分辨率减少并且预先过滤的纹理图。对于一个  $8 \times 8$  的纹理来说，需要为它构造  $4 \times 4$ 、 $2 \times 2$ 、 $1 \times 1$  这三个 `mipmap`。如果正方形被缩小为在屏幕上占  $6 \times 6$  的像素矩阵，一个像素的采样过程就变成这样，先到  $8 \times 8$  的纹理图中对最接近它的  $2 \times 2$  的纹理单元矩阵进行采样（也就是上面的线性过滤）；然后到  $4 \times 4$  的纹理图中重复上面的过程；接着把上面两次采样的结果进行加权平均，得到最后的采样数据。可以看出整个过程一共进行了 3 次线性过滤，所以这种方法叫做三线性过滤，它的效果是三种纹理过滤方式中面最好的。

接下来看一下 `handleLoadedTexture` 函数中的代码。在该函数中，同时支持 3 种纹理过滤方式的使用，代码如下所示。

```
function handleLoadedTexture(textures)
{
    // 通知 WebGL 上下文对象将纹理图像垂直翻转
    gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);
    // 通知 WebGL 上下文对象 texture[0] 为当前所使用的纹理对象
```

```

gl.bindTexture(gl.TEXTURE_2D, textures[0]);
// 将纹理图像传入显卡
gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE,
textures[0].image);
// 使用最近点采样方式
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST);
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.NEAREST);
// 通知 WebGL 上下文对象 texture[1] 为当前所使用的纹理对象
gl.bindTexture(gl.TEXTURE_2D, textures[1]);
// 将纹理图像传入显卡
gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE,
textures[1].image);
// 使用线性过滤方式
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR);
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
// 通知 WebGL 上下文对象 texture[2] 为当前所使用的纹理对象
gl.bindTexture(gl.TEXTURE_2D, textures[2]);
// 将纹理图像传入显卡
gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE,
textures[2].image);
// 使用三线性过滤方式
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR);
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER,
gl.LINEAR_MIPMAP_NEAREST);
// 创建纹理图像的 mipmap
gl.generateMipmap(gl.TEXTURE_2D);
}

```

在这段代码中，同时对一个数组中的三个纹理对象进行参数设定工作，第 1 个纹理对象使用最近点采样纹理过滤方式，第 2 个纹理对象使用线性过滤方式，第 3 个纹理对象使用三线性过滤方式，这样在绘制图像时，根据用户选择的线性过滤方式，使用对应的纹理对象进行绘制即可。

接下来看一下创建纹理对象数组的 `initTexture` 函数，代码如下所示。

```

var crateTextures = Array(); // 纹理对象数组
// 创建纹理对象
function initTexture()
{
    var crateImage = new Image();
    for (var i=0; i < 3; i++)
    {
        var texture = gl.createTexture();
        texture.image = crateImage;
        crateTextures.push(texture);
    }
    crateImage.onload = function() {
        handleLoadedTexture(crateTextures)
    }
    crateImage.src = "create.gif";
}

```

在这段代码中，创建了一个纹理对象的数组，数组中存放三个纹理对象，对这三个纹理对象指定相同的纹理图像，并且在纹理图像装载完毕后调用 `handleLoadedTexture` 函数来设置这三个纹理对象的参数，分别指定三个纹理对象使用不同的纹理过滤方式。

现在通过绘制图像的 `drawScene` 函数来具体看一下这三个纹理对象是如何被使用的，以及用户在按下按键时立方体的旋转速度与旋转角度是如何被改变的。

本示例程序中的 `drawScene` 函数在 12.6 节示例程序的 `drawScene` 函数的基础上作出三个改变，首先在绘制立方体时，并不是直接指定其在 Z 轴上的绘制位置，而是使用变量来指定，代码如下所示。

```
mat4.translate(mvMatrix, [0.0, 0.0, z]);
```

接下来，将如下三行代码中第 3 行代码删除，使立方体不在 Z 轴上进行旋转。

```
mat4.rotate(mvMatrix, degToRad(xRot), [1, 0, 0]);
mat4.rotate(mvMatrix, degToRad(yRot), [0, 1, 0]);
mat4.rotate(mvMatrix, degToRad(zRot), [0, 0, 1]);
```

最后，在绘制立方体的时候，指定应该使用纹理对象数组中的哪一个纹理对象来进行绘制，代码如下所示。

```
gl.bindTexture(gl.TEXTURE_2D, crateTextures[filter]);
```

用户每次按下 F 键时，程序都会修改变量 `filter` 的值，使其在 0、1 和 2 这 3 个值中进行变动，从而实现纹理过滤方式的改变。

`drawScene` 函数的完整代码如下所示。

```
// 绘制立方体
function drawScene()
{
    // 设置 3D 视图的视图大小
    gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
    // 擦除 canvas 元素中内容
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
    // 设置对视图的观察视角
    mat4.perspective(45, gl.viewportWidth / gl.viewportHeight, 0.1, 100.0,
        pMatrix);
    // 使用恒等矩阵进行初始化，将绘制位置设置在视图中央
    mat4.identity(mvMatrix);
    // 将绘制位置后移 5 个单位
    mat4.translate(mvMatrix, [0.0, 0.0, z]);
    // 旋转立方体
    mat4.rotate(mvMatrix, degToRad(xRot), [1, 0, 0]);
    mat4.rotate(mvMatrix, degToRad(yRot), [0, 1, 0]);
    // 将 cubeVertexPositionBuffer 设定为接下来的操作所使用的缓冲区
    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
    /* 通知 WebGL 上下文对象缓冲区中存放的顶点位置信息将作为立方体各顶点位置信息来使用，每个立方体使用三个数据（三维信息）*/
```

```

gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
cubeVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);
// 将 cubeVertexTextureCoordBuffer 设定为接下来的操作所使用的缓冲区
gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexTextureCoordBuffer);
/* 通知 WebGL 上下文对象缓冲区中存放的顶点纹理信息将作为立方体各顶点纹理信息来使用 */
gl.vertexAttribPointer(shaderProgram.textureCoordAttribute,
cubeVertexTextureCoordBuffer.itemSize, gl.FLOAT, false, 0, 0);
// 通知 WebGL 上下文对象最先处理的纹理为 TEXTURE0
gl.activeTexture(gl.TEXTURE0);
gl.bindTexture(gl.TEXTURE_2D, crateTextures[filter]);
// 设置程序对象中的一致变量的值为 0
gl.uniform1i(shaderProgram.samplerUniform, 0);
// 将 cubeVertexIndexBuffer 设定为接下来的操作所使用的缓冲区
gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVertexIndexBuffer);
// 通知 WebGL 上下文对象使用当前的模型 - 视图矩阵与投影矩阵
setMatrixUniforms();
// 绘制立方体
gl.drawElements(gl.TRIANGLES, cubeVertexIndexBuffer.numItems,
gl.UNSIGNED_SHORT, 0);
}

```

最后，对 `animate` 函数进行一些小的修改，不再设定立方体的旋转速度为固定速度，而是使用户通过按键来修改的变量 `xSpeed` 与变量 `ySpeed` 以设定立方体的旋转速度。修改完毕后该函数中的代码如下所示。

```

// 设置动画参数
function animate()
{
    // 获取当前时间
    var timeNow = new Date().getTime();
    // 如果动画绘制已经开始
    if (lastTime != 0)
    {
        // 获取当前时间与上次动画绘制时间的间隔时间
        var elapsed = timeNow - lastTime;
        xRot += (xSpeed * elapsed) / 1000.0;
        yRot += (ySpeed * elapsed) / 1000.0;
    }
    lastTime = timeNow;
}

```

接下来还需要添加一些处理用户按键事件的代码，使用户可以通过按键来修改纹理过滤方式、立方体旋转速度以及旋转角度。

首先需要在 `webGLStart` 函数中添加如下两行代码。

```

// 利用函数来处理用户按键事件
document.onkeydown = handleKeyDown;
document.onkeyup = handleKeyUp;

```

很明显，这两行代码的含义为当用户按下按键时调用 JavaScript 的 `handleKeyDown` 函数，当用户松开按键时调用 JavaScript 的 `handleKeyUp` 函数。

修改完毕后的 `webGLStart` 函数的代码如下所示。

```
// 绘制 3D 图形
function webGLStart()
{
    var canvas = document.getElementById("canvas1");// 获取 canvas 元素
    initGL(canvas);// 初始化 WebGL 上下文对象
    initShaders();// 初始化渲染器
    initBuffers();// 初始化缓冲区
    initTexture();// 初始化纹理
    // 每次清除 canvas 元素中的内容时均将其填充为黑色
    gl.clearColor(0.0, 0.0, 0.0, 1.0);
    gl.enable(gl.DEPTH_TEST);// 使用深度测试
    // 利用函数来处理用户按键事件
    document.onkeydown = handleKeyDown;
    document.onkeyup = handleKeyUp;
    tick();// 调用 tick 函数
}
```

接下来看一下 `handleKeyDown` 函数与 `handleKeyUp` 函数的代码。

```
var currentlyPressedKeys = {};
function handleKeyDown(event)
{
    currentlyPressedKeys[event.keyCode] = true;
    if (String.fromCharCode(event.keyCode) == "F")
    {
        filter += 1;
        if (filter == 3)
            filter = 0;
    }
}
function handleKeyUp(event)
{
    currentlyPressedKeys[event.keyCode] = false;
}
```

很明显，`handleKeyDown` 函数所作的处理就是每次用户按下 F 键时修改变量 `filter` 的值，使其在 0、1 和 2 这 3 个值中进行变动，因此在 `drawScene` 函数中可以根据这个变量 `filter` 的值来改变纹理的过滤方式。

另外，使用了一个 JavaScript 中的伪哈希表。所谓 JavaScript 中的伪哈希表，是指通过一个给定的数字键名，可以返回与该按键名对应的键值。在本示例中，通过给定的用户按键的键盘编号 (keycode)，返回该用户是否按下该按键的布尔值 (True 或 False)。例如，当用户按下 PageUp 键时，因为 PageUp 键的键盘编号为 33，所以 `currentlyPressedKeys[33]` 被设为 True，当用户松开 PageUp 键时，`currentlyPressedKeys[33]` 被设为 False。



这里解释一下为什么需要使用这个伪哈希表来保存每个按键的状态。

在一个计算机游戏或需要处理用户按键的三维图像系统中，用户按键事件可分为两种不同的情况：

1) 用户迅速按下按键之后迅速放开。例如射击游戏中的“开枪”，拳击游戏中的“出拳”，用户都是按下按键之后立即放开。

2) 用户按住按键后并不立即放开。例如动作游戏中的“走路”，用户要按住按键一直到游戏人物走到目的地为止。

对于情况 2)，用户在按住某个按键的同时，往往还要按下第二个按键，从而希望同时处理两个按键事件。例如在动作游戏中，“走路”的同时还希望使游戏人物呈快跑状态，或者让游戏人物往前走的同时能够往上跳一下（如碰到了某个台阶）。

在这里，用户按一下 F 键，符合上述的情况 1)。用户按下 F 键后立刻改变纹理的过滤方式，每按一次都将纹理的过滤方式改变一次。而用户按下其他按键时符合情况 2)，例如，用户按住方向键不放，则随着按下按键的时间越来越长，旋转的速度也越来越快。同时能够在按住方向键的同时，按下 PageUp 键或 PageDown 键来进行图像的拉近或拉远（或者说进行图像的缩放）操作。

事实上，对于情况 2)，可以使用单独的函数 `handleKeys` 来进行处理，稍后会讲到这个函数。在这之前，先来看一下调用该函数的 `tick` 函数的代码，如下所示。

```
function tick()
{
    requestAnimationFrame(tick);
    handleKeys();
    drawScene();
    animate();
}
```

接下来看一下 `handleKeys` 函数的代码，如下所示。

```
function handleKeys()
{
    if (currentlyPressedKeys[33]) // 按下 PageUp 键
        z -= 0.05; // 拉近图像（图像被放大）
    if (currentlyPressedKeys[34]) // 按下 PageDown 键
        z += 0.05; // 拉远图像（图像被缩小）
    if (currentlyPressedKeys[37]) // 按下左方向键
        /* 改变 Y 轴上的旋转速度与旋转方向（水平方向旋转，值越小则旋转得越慢，变成 0 为不旋转，变成负数为反方向旋转，值越小则旋转得越快）*/
        ySpeed -= 1;
    if (currentlyPressedKeys[39]) // 按下右方向键
        /* 改变 Y 轴上的旋转速度与旋转方向（水平方向旋转，值越大则旋转得越快，变成 0 为不旋转，变成负数为反方向旋转，值越大则旋转得越慢）*/
        ySpeed += 1;
    if (currentlyPressedKeys[38]) // 按下向上方向键
        /* 改变 X 轴上的旋转速度与旋转方向（垂直方向旋转，值越小则旋转得越慢，变成 0 为不旋
```

```

        转，变成负数为反方向旋转，值越小则旋转得越快)*/
        xSpeed -= 1;
    if (currentlyPressedKeys[40]) // 按下向下方向键
        /* 改变 X 轴上的旋转速度与旋转方向（垂直方向旋转，值越大则旋转得越快，变成 0 为不旋
        转，变成负数为反方向旋转，值越大则旋转得越慢）*/
        xSpeed += 1;
}

```

由于在动画中，每次绘制完图像后都会重新调用 tick 函数，tick 函数调用 handleKeys 函数。在 handleKeys 函数中，首先检查某个按键是否处于按下状态，并且根据该按键的状态来设置全局变量 z，xSpeed 与 ySpeed，最终实现根据用户按键状态来改变动画中立方体的 Z 轴方向上的位置。

## 12.8 本章小结

本章对 HTML 5 中的 WebGL API 进行了一个初步的介绍，包括 WebGL API 的运行环境的准备，如何使用 WebGL API 绘制彩色的三维图形与图像、制作三维动画、对三维图像使用纹理以及如何让用户通过键盘来对三维场景中的图形、图像进行控制。希望通过对本章内容的阅读，读者能够对 WebGL API 有了一个基本了解，能够在自己的网页中绘制出符合需要的三维图形、图像与动画。

## 附录 五大浏览器的最新版对 HTML5 的支持情况

附表 1 ~附表 5 列出了截至 2011 年 7 月五大浏览器的最新版对 HTML 5 的支持情况。

附表 1 对全局属性的支持（√表示支持，×表示不支持）

属性	Firefox 5.0	Opera 11.50	Chrome 12.0	Safari 5.0	IE 9.0
hidden 全局属性	√	√	√	√	√
contenteditable 全局属性	√	√	√	√	√

附表 2 对表单元素的支持（√表示支持，×表示不支持）

元素	Firefox 5.0	Opera 11.50	Chrome 12.0	Safari 5.0	IE 9.0
input type=search	√	√	√	√	×
input type=tel	√	√	√	√	×
input type=url	√	√	√	√	×
input type=email	√	√	√	√	×
input type=datetime	×	√	√	不支持特殊用户界面，支持 min 属性、max 属性和 step 属性	×
input type=date	×	√	√	不支持特殊用户界面，支持 min 属性、max 属性和 step 属性	×
input type=month	×	√	√	不支持特殊用户界面，支持 min 属性 max 属性和 step 属性	×
input type=week	×	√	√	不支持特殊用户界面，支持 min 属性、max 属性与 step 属性	×
input type=time	×	√	√	不支持特殊用户界面，支持 min 属性、max 属性和 step 属性	×
input type=datetime-local	×	√	√	不支持特殊用户界面，支持 min 属性、max 属性和 step 属性	×
input type=number	仅支持 min 属性与 max 属性，不支持特殊用户界面，提交时验证和 step 属性	√	√	√	×

(续)

元素	Firefox 5.0	Opera 11.50	Chrome 12.0	Safari 5.0	IE 9.0
input type=range	仅支持 min 与 max 属性, 不支持特殊用户界面、提交时验证和 step 属性	√	√	√	×
input type=color	×	支持特殊用户界面, 不支持提交时验证	不支持特殊用户界面, 支持提交时验证	不支持特殊用户界面, 支持提交时验证	×
input type=checkbox 元素的 indeterminate 属性	√	×	√	√	×
Select 元素的 required 属性	√	×	√	×	×
datalist	√	√	×	×	×
output	√	√	√	×	×
progress	×	√	√	×	×
meter	×	√	√	×	×

附表 3 对其他元素的支持 (√ 表示支持, × 表示不支持)

元素	Firefox 5.0	Opera 11.50	Chrome 12.0	Safari 5.0	IE9.0
新增结构元素	√	√	√	√	√
figure 元素	√	√	√	×	√
figcaption 元素	√	√	√	×	√
mark 元素	√	√	√	×	√
time 元素	×	×	×	×	×
details 元素	×	×	√	×	×
summary 元素	√	√	√	√	×
command 元素	×	×	×	×	×
menu 元素	√	√	√	√	×

附表 4 对表单属性的支持 (√ 表示支持, × 表示不支持)

属性	Firefox 5.0	Opera 11.50	Chrome 12.0	Safari 5.0	IE 9.0
提交时验证的 pattern 属性	√	√	√	√	×
提交时验证的 required 属性	√	√	√	√	×
form 属性	√	√	√	×	×
formaction 属性	√	√	√	×	×
formmethod 属性	√	√	√	×	×
formnovalidate 属性	√	√	√	○	×
autocomplete 属性	√	√	×	×	○

(续)

属性	Firefox 5.0	Opera 11.50	Chrome 12.0	Safari 5.0	IE 9.0
autofocus 属性	√	√	√	√	×
placeholder 属性	√	√	√	√	×
multiple 属性	√	√	√	√	×
CSS 选择器	√	√	√	√	√

附表 5 对 API 的支持 (√表示支持, ×表示不支持)

属性	Firefox 5.0	Opera 11.50	Chrome 12.0	Safari 5.0	IE 9.0
canvas 元素	√	√	√	√	√
Canvas 2D API	√	√	√	√	√
Canvas 3D API (现已改为 WebGL 3D API)	√	√	√	×	√
Canvas Text	√	√	√	√	√
video 元素	√	√	√	√	√
MPEG-4 支持	×	×	×	√	√
H-264 支持	×	×	√	√	√
Ogg Theora 支持	√	√	√	×	×
WebM 支持	√	√	√	×	√
audio 元素	√	√	√	√	√
PCM audio 支持	√	√	√	√	×
MP3 支持	×	×	√	√	√
AAC 支持	×	×	√	√	√
Ogg Vorbis 支持	√	√	√	×	×
WebM 支持	√	√	√	×	√
拖放 API	√	×	√	√	√
Geolocation API	√	√	√	√	√
跨域通信	√	√	√	√	√
WebSocket	√ (在最新版浏览器中, 因为在协议上存在基础安全隐患, 所以被禁用, 一旦协议更新, 将恢复使用)	√ (在最新版浏览器中, 因为在协议上存在基础安全隐患, 所以被禁用, 一旦协议更新, 将恢复使用)	√	√	×
文件 API——FileReader API	√	×	√	×	×
文件 API——FileWriter API	×	×	×	×	×
Session Storage	√	√	√	√	√
Local Storage	√	√	√	√	√
Web SQL Database	×	√	√	√	×
IndexedDB	√	×	√	×	×
Web Workers	√	√	√	√	×



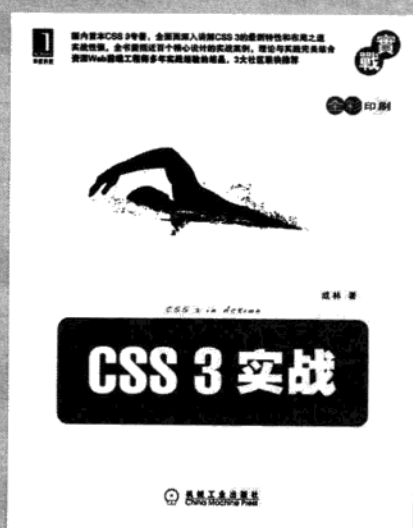
书号: 978-7-111-35873-2  
定价: 59.00元



书号: 978-7-111-33624-2  
定价: 69.00元



书号: 978-7-111-32543-7  
定价: 59.00元



书号: 978-7-111-34155-0  
定价: 69.00元



专业成就人生  
立体服务大众

www.hzbook.com

## 填写读者调查表 加入华章书友会 获赠精彩技术书 参与活动和抽奖

尊敬的读者：

感谢您选择华章图书。为了聆听您的意见，以便我们能够为您提供更优秀的图书产品，敬请您抽出宝贵的时间填写本表，并按底部的地址邮寄给我们（您也可通过www.hzbook.com填写本表）。您将加入我们的“华章书友会”，及时获得新书资讯，免费参加书友会活动。我们将定期选出若干名热心读者，免费赠送我们出版的图书。请一定填写书名书号并留全您的联系信息，以便我们联络您，谢谢！

书名：

书号：7-111-( )

姓名：	性别： <input type="checkbox"/> 男 <input type="checkbox"/> 女	年龄：	职业：
通信地址：		E-mail：	
电话：	手机：	邮编：	

### 1. 您是如何获知本书的：

☐ 朋友推荐 ☐ 书店 ☐ 图书目录 ☐ 杂志、报纸、网络等 ☐ 其他

### 2. 您从哪里购买本书：

☐ 新华书店 ☐ 计算机专业书店 ☐ 网上书店 ☐ 其他

### 3. 您对本书的评价是：

技术内容	<input type="checkbox"/> 很好	<input type="checkbox"/> 一般	<input type="checkbox"/> 较差	<input type="checkbox"/> 理由_____
文字质量	<input type="checkbox"/> 很好	<input type="checkbox"/> 一般	<input type="checkbox"/> 较差	<input type="checkbox"/> 理由_____
版式封面	<input type="checkbox"/> 很好	<input type="checkbox"/> 一般	<input type="checkbox"/> 较差	<input type="checkbox"/> 理由_____
印装质量	<input type="checkbox"/> 很好	<input type="checkbox"/> 一般	<input type="checkbox"/> 较差	<input type="checkbox"/> 理由_____
图书定价	<input type="checkbox"/> 太高	<input type="checkbox"/> 合适	<input type="checkbox"/> 较低	<input type="checkbox"/> 理由_____

### 4. 您希望我们的图书在哪些方面进行改进？

---



---

### 5. 您最希望我们出版哪方面的图书？如果有英文版请写出书名。

---



---

### 6. 您有没有写作或翻译技术图书的想法？

☐ 是，我的计划是\_\_\_\_\_ ☐ 否

### 7. 您希望获取图书信息的形式：

☐ 邮件 ☐ 信函 ☐ 短信 ☐ 其他\_\_\_\_\_

请寄：北京市西城区百万庄南街1号 机械工业出版社 华章公司 计算机图书策划部收  
邮编：100037 电话：(010) 88379512 传真：(010) 68311602 E-mail: hzjsj@hzbook.com

[ General Information ]

书名=HTML 5开发精要与实例详解

作者=陆凌牛著

页数=550

出版社=北京市：机械工业出版社

出版日期=2012.01

SS号=12947100

DX号=000008201641

URL=<http://book.szdnet.org.cn/bookDetail.jsp?dxNumber=000008201641&d=9D422DFF3F8148B155EC0FD615E1517F>